எளிய இனிய கணினி மொழி

ரூபி



பிரியா சுந்தரமூர்த்தி

எளிய இனிய கணினி மொழி - ரூபி

பிரியா சுந்தரமூர்த்தி

எளிய இனிய கணினி மொழி - ரூபி

முதல் பதிப்பு ஜனவரி 2016

ஆசிரியர் - பிரியா சுந்தரமூர்த்தி priya.budsp@gmail.com

பதிப்பாசிரியர், பிழை திருத்தம், வடிவமைப்பு : இல. கலாராணி lkalarani@gmail.com

அட்டைப்படம் - மனோஜ் குமார் - socrates1857@gmail.com

நூல் மூலம் :

[*http://static.kaniyam.com/ebooks/learn-ruby-in-tamil/learn-ruby-in-tamil.odt*](http://static.kaniyam.com/ebooks/learn-ruby-in-tamil/learn-ruby-in-tamil.odt)

அறிமுகவுரை

நான் M.C.A படித்து விட்டு நான்கு வருடங்களாக பணியில் இருந்தேன். எனது திருமணத்துக்கு பின்பு பணியை தொடர இயலவில்லை. ஆனால், பணியை தொடர முடியவில்லை என்ற வருத்தம் இருந்தது. வீட்டிலிருந்த படியே எதுவும் செய்ய முடியுமா என்ற எண்ணம் உறுத்திக்கொண்டே இருந்தது. ஒரு வழியாக, வீட்டிலிருந்தபடி படிக்கலாம் என்ற முடிவோடு கணியம் ஆசிரியர் உதவியில் படிக்க நினைத்தேன். அப்பொழுது அவரிடம், "எளிய தமிழில் ரூபி" இல்லையா? என்று கேட்டேன். அதற்கு அவர், "இல்லை, ஏன் நீங்கள் எழுத முயற்சி செய்யுங்கள்" என்றார். நான் சிறிது அதிர்ச்சியடைந்து, "என்னால் முடியுமா" என்றேன். "கண்டிப்பாக முடியும் முயற்சியுங்கள்" என்றார். அந்த ஊக்கமே இன்று இதை எழுத உதவியது.

இது படிப்பவருக்கு எளிதாகவும், உதவியாகவும் இருக்குமென்று நம்புகிறேன். உங்கள் கருத்துகளை ஆவலுடன் எதிர்பார்க்கிறேன்.

நன்றி.

பிரியா சுந்தரமூர்த்தி
priya.budsp@gmail.com

ஐக்கிய அரபு அமீரகம்

பதிப்பாசிரியர் உரை

"எளிய இனிய கணினி மொழி" - ரூபிக்கு இதை விட பொருத்தமான விளக்கத்தை அளித்திருக்க முடியாது. இன்று பெரும்பாலான இணைய பயன்பாடுகள் ரூபியில் எழுதப்படுகின்றன. நிரலை சுருக்கமாக எழுதுவதே ரூபியின் சக்திவாய்ந்த அம்சங்களில் ஒன்றாகும். மென்பொருட்களை அதிவிரைவாகவும், எளிமையாகவும் ரூபியில் உருவாக்க முடியும். ரூபியின் அடிப்படையையும், பரவலாக பயன்படுத்தப்படும் அம்சங்களையும் பிரியா இந்நூலில் விவரித்திருக்கிறார். ரூபியின் எளிமையும், இனிமையும் இந்நூலெங்கும் வியாபித்திருப்பது அவரது சிறப்பு.

"பிறநாட்டு நல்லறிஞர் சாத்திரங்கள்
தமிழ்மொழியிற் பெயர்த்தல் வேண்டும்;
இறவாத புகழுடைய புதுநூல்கள்
தமிழ்மொழியில் இயற்றல் வேண்டும்;"

பாரதியின் இக்கனவினை மெய்ப்பிக்கும் முயற்சியில் கணியம் 2012 முதல் ஈடுபட்டிருக்கிறது. கட்டற்ற மென்பொருட்களைப்பற்றிய தகவல்களையும், மென்பொருள் உருவாக்க முறைகளப்பற்றியும்

தொடர்கட்டுரைகள் கணியம் இதழில், தமிழில் வெளியாகி வருகின்றன. இதில் ரூபி என்ற நிரலாக்க மொழியை பற்றிய பதிவுகளைத் தொகுத்து இந்த மின்னூலை வெளியிடுகிறோம்.

உங்கள் கருத்துகளையும், பிழைதிருத்தங்களையும் editor@kaniyam.com என்ற முகவரிக்கு எழுதுங்கள்.

படிப்போம்! பகிர்வோம்!! பயன் பெறுவோம்!!!

நன்றி,

இல. கலாராணி
lkalarani@gmail.com

பொருளடக்கம்

1 ரூபியின் வரலாறு

ரூபி ஒரு எளிமையாக புரிந்துகொள்ளக்கூடிய பொருள் நோக்கு நிரலாக்க மொழி *(object oriented programming language). 1993*, ஜப்பானில் யூகிஹிரோ மேட்சுமோடோ ***(Yukihiro Matsumoto)*** என்பவரால் ரூபி உருவாக்கப்பட்டது. அவரை அன்பாக மேட்ஸ் *(Matz)* என்றும் அழைப்பர். *1995*-ம் ஆண்டில் தனது நாடான ஜப்பானில் இவர் ரூபியை அறிமுகப்படுத்தினார். *2000*-ம் ஆண்டு தொடக்கத்தில் அனைத்து நாடுகளில் உள்ள நிரலாக்க உலகத்தவரால் சிறந்த பொருள் நோக்கு நிரலாக்க மொழியாக, ரூபி அங்கீகரிக்கப்பட்டது.

1.1 ரூபி என்றால் என்ன?

ரூபி ஒரு object oriented interpreted scripting language. C, C++, ஜாவா, C# போன்ற மொழிகளில் ஒரு நிரலை (source code) எழுதிய பின், அதை செயல்படுத்தி பார்க்கும் முன்னதாக, அதனை அடிநிலைமொழிக்கு பெயர்க்க வேண்டும் (compile). ரூபி போன்ற interpreted மொழிகளில், இந்த இடைபட்டநிலை இல்லை. நிரல் செயல்படுத்தப்படும் பொழுது மட்டுமே ஆணைபெயர்ப்பியால் (interpreter) ஒவ்வொரு படியாக அடிநிலை மொழிக்கு பெயர்க்கப்படும்.

Interpreted மொழிகளில் நன்மைகள் மற்றும் தீமைகள் கலந்தே உள்ளது. இதனுடைய முதன்மையான சிறப்பு என்னவென்றால் interpreted மொழியானது பலதரப்பட்ட இயங்குதளம் (operating system platform) மற்றும் வன்பொருள் கட்டமைப்புகளில் (hardware architectures) முழுவதுமாக இயங்கவல்லது. ஆனால், நிரல்பெயர்க்கப்பட்ட பயன்பாடுகளானது (compiled application) எந்த இயங்குதளம் மற்றும் வன்பொருளுக்காக மொழிபெயர்க்கப்பட்டதோ அதில் மட்டுமே இயங்கும். மற்றுமொரு சிறப்பு என்னவென்ரால் ரூபி ஆணைபெயர்ப்பியில், நிகழ்நேரத்தில் (real time) ரூபி நிரலை எழுதி இயக்க முடியும்.

இதனுடைய முதன்மையான பின்னடைவு இதனுடைய வேகம். ஏனென்றால் மூலநிரலானது இயங்குநேரத்தில் *(runtime)* பெயர்க்கப்படுகிறது *(interpret)*. நிரல்பெயர்க்கப்பட்ட பயன்பாடுகளுடன் *(Compiled application)* ஒப்பிடுகையில் இது சற்று மெதுவாகவே செயல்படும். அடுத்தப்படியாக, *interpreted* மொழியில் உருவான பயன்பாடுகளை உபயோகிப்பவரால் அதன் மூலநிரலைக்காண இயலும்.

2 ரூபி நிறுவுதல்:

ரூபி C மொழியில் எழுதப்பட்டது. எனவே அது operating system மற்றும் hardware platform க்குப் பொருத்தமான binary distribution-னை தேர்வு செய்து நிறுவ வேண்டும், அல்லது ரூபி மூலநிரலை பதிவிறக்கி, compile செய்து நிறுவ வேண்டும்.

2.1Red Hat Enterprise மற்றும் Fedora Linux-ல் ரூபி நிறுவுதல்:

Red Hat Enterprise மற்றும் Fedora Linux இரண்டும் YUM installation Manager மற்றும் rpm-யை பயன்படுத்துகின்றன. முதலவதாக ரூபி ஏற்கனவே நிறுவப்பட்டுள்ளதா என்று பார்க்க வேண்டும். இதற்கு rpm command-டை பயன்படுத்தலாம். பின்வரும் எடுத்துக்காட்டில் ruby இன்னும் install செய்யப்படவில்லை.

```
rpm -q ruby
package ruby is not installed
```

ரூபி நிறுவப்பட படவில்லையெனில், yum update manager-ரை உபயோகப்படுத்தி நிறுவ முடியும். இதை root-டை

கொண்டு செய்ய வேண்டும். ஆதலால் இதற்கு super user password தேவை.

```
su -
yum install ruby
```

yum tool ஆனது, ruby package மற்றும் இதர packages-களையும் நிறுவி விடும்.

```
Downloading Packages:
(1/2): ruby-1.8.1-7.EL4.8 100% |                |
156 kB    00:10
(2/2): ruby-libs-1.8.1-7. 100% |                |
1.5 MB    01:23
Running Transaction Test
Finished Transaction Test
Transaction Test Succeeded
Running Transaction
  Installing: ruby-libs
######################## [1/2]
  Installing: ruby
######################## [2/2]

Installed: ruby.i386 0:1.8.1-7.EL4.8
Dependency Installed: ruby-libs.i386
0:1.8.1-7.EL4.8
Complete!
```

ரூபி நிறுவப்பட்டவுடன், மேற்குறிப்பிட்ட rpm command-டை பயன்படுத்தி அதனை சரிப்பார்க்கலாம்.

```
rpm -q ruby
ruby-1.8.1-7.EL4.8
```

மாறாக, ரூபி install ஆனதை command line-ல் run செய்து பதிப்பு (version) விவரங்களை பெறலாம்.

```
ruby -v
ruby 1.8.1 (2003-12-25) [i386-linux-gnu]
```

2.2ubuntu மற்றும் debian linux-ல் ரூபி installation:

apt-get கருவியை உபயோகப்படுத்தி Debian, Ubuntu மற்றும் மற்ற debian derived linux distrubutions-ல் நிறுவ முடியும். நீங்கள் Ubuntu linux பயன்படுத்தினால், பின்வருமாறான output ruby command மூலம் பெறலாம்.

```
$ ruby
The program 'ruby' is currently not
installed.  You can install it by typing:
sudo apt-get install ruby
-bash: ruby: command not found
```

ரூபியை நிறுவ, *apt-get command*-டை இயக்கலாம்.

```
sudo apt-get install ruby
```

apt-get tool ஆனது ரூபி சார்ந்த மற்ற *package*-களையும் நிறுவி விடும்.

```
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following extra packages will be
installed:
  libruby1.8 ruby1.8
Suggested packages:
  ruby1.8-examples rdoc1.8 ri1.8
The following NEW packages will be
installed:
  libruby1.8 ruby ruby1.8
0 upgraded, 3 newly installed, 0 to remove
and 135 not upgraded.
Need to get 1769kB of archives.
After unpacking 6267kB of additional disk
space will be used.
Do you want to continue [Y/n]?
```

Installation முடிவடைந்ததும், ரூபி நிறுவப்பட்டுள்ளதா என்பதை *command line* மூலம் ரூபி *version* உடன் அறிந்து

கொள்ளலாம்.

```
ruby -v
ruby 1.8.1 (2003-12-25) [i386-linux-gnu]
```

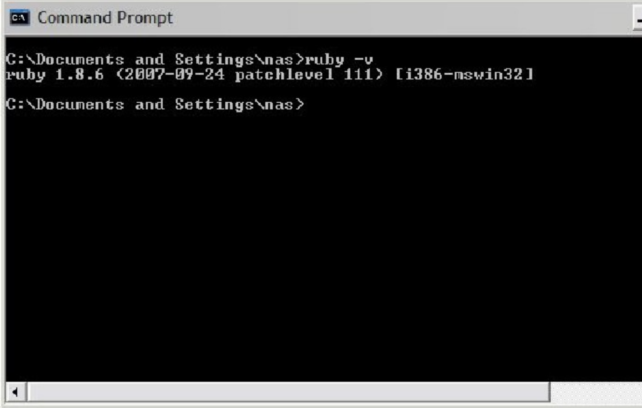
2.3Microsoft Windows-ல் ரூபி installation:

ரூபியை *windows*-இல் நிறுவும் எளிமையான வழியை *one-click ruby installer* எனலாம். இது ஒரு *executable*. இது ரூபியை எளிமையாக நீக்கும் *mechanism* கொண்டது.

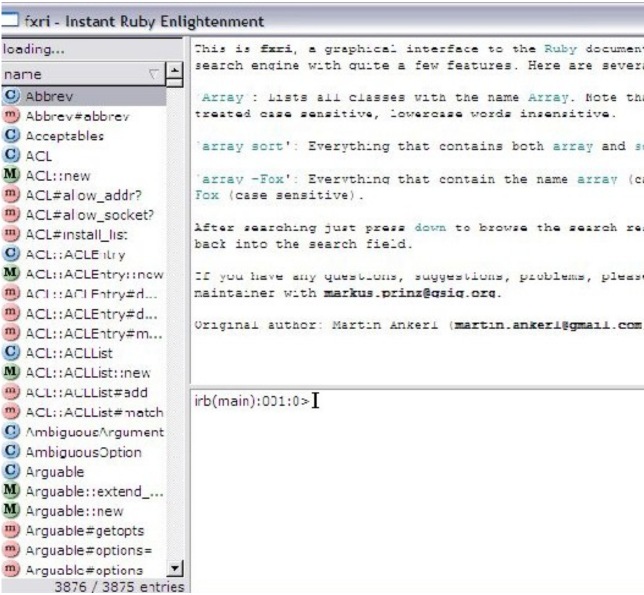
One-click ruby Installer-ரை உபயோகிக்க *http://rubyforge.org* சென்று *one-click Installer*-ரை *click* செய்யவும். இரண்டாவது பக்கத்தில், பல்வேறு *one-click Installer release*-கள் பட்டியலிடப்படும். *Executable* பதிவிறக்கம் *(download)* செய்த பிறகு அதை மற்ற *windows application* போல *launch* செய்யலாம்.

ரூபியை நிறுவ நிறைய வழிகள் உண்டு. அதில் மிகவும் அடிப்படையான அணுகுமுறை, *windows command prompt*-ல் ரூபியை இயக்கலாம்.

Install ஆன ரூபியின் *version*-னை பின்வருமாறு *system*-ல் காணலாம்.



மற்றொன்று, *fxri tool*-லை *windows start menu*-விலிருந்து *launch* செய்யலாம். இது ஒரு *interactive tool*, இது *ruby documentation*-க்கும் மற்றும் *ruby console*-க்கும் *access*-ஐ வழங்கும்.


fxri - Instant Ruby Enlightenment
loading...
name
Abbrev
Abbrev#abbrev
Acceptables
ACL
ACL::new
ACL#allow_addr?
ACL#allow_socket?
ACL#install_list
ACL::ACLEntry
ACL::ACLEntry::new
ACL::ACLEntry#d...
ACL::ACLEntry#d...
ACL::ACLEntry#m...
ACL::ACLList
ACL::ACLList::new
ACL::ACLList#add
ACL::ACLList#match
AmbiguousArgument
AmbiguousOption
Arguable
Arguable::extend_...
Arguable::new
Arguable#getopts
Arguable#options=
Arguable#options
3876 / 3875 entries
This is fxri, a graphical interface to the Ruby document
search engine with quite a few features. Here are severa
'Array': Lists all classes with the name Array. Note tha
treated case sensitive, lowercase words insensitive.
'array sort': Everything that contains both array and s
'array -Fox': Everything that contain the name array (ca
Fox (case sensitive).
After searching just press down to browse the search re
back into the search field.
If you have any questions, suggestions, problems, please
maintainer with markus.prinz@qsig.org.
Original author: Martin Ankerl (martin.ankerl@gmail.com
irb(main):001:0>

3 எளிய ரூபி எடுத்துக்காட்டுகள்:

ரூபி ஒரு எளிமையான scripting language ஆகும். இதன் syntax-ம் மிகவும் எளிமையானது. அழகானது. Programming உலகில் பாரம்பரியமாக முதல் எடுத்துக்காட்டாக பயன்படுத்துவது "hello world" ஐ print செய்வதாகும். ஆனால் இதில் சிறு மாற்றமாக "Hello Ruby" என print செய்யலாம்.

GNU/Linux ல்,

```
print "Hello Ruby!\n"
```

windows-ல்

```
print "Hello Ruby!"
```

சில வார்த்தைகளை output ஆக பெற ஒரு வரி ரூபி நிரலே போதுமானது. நாம் முந்தைய அத்தியாயத்தில் பார்த்தப்படி, ரூபியின் பலம், அதனின் வேகமும், எளிமையான முறையில் கற்கவல்லதும்தான். இந்த எடுத்துக்காட்டுக்கு சமமான Java நிரலுடன் ஒப்பிடலாம்.

```
import java.io.*;
```

```
public class Hello {

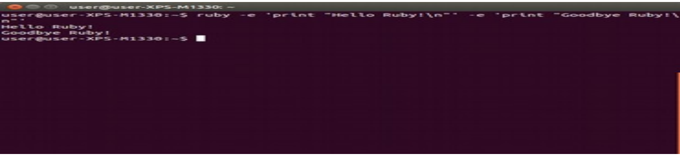
        public static void main(String[]
args)
        {
                System.out.println("Hello
Ruby!\n");
        }
}
```

Java போன்ற programming language-ல் எளிய பணியை செய்யக்கூட நிறைய நிரல் எழுத வேண்டும். ஆனால் ரூபியில் print-டை தொடர்ந்து output string-ஐக் கொடுத்தால் போதுமானது.

ரூபி நிரலை இயக்க (execute) பல வழிகள் உள்ளன. அவற்றைப் பின்வருமாறு காணலாம்.

3.1Command line-ல் ரூபியை execute செய்தல்:

```
ruby -e 'print "Hello Ruby!\n"' -e 'print
"Goodbye Ruby!\n"'
Hello Ruby!
Goodbye Ruby!
```



நமது *"Hello Ruby"* எடுத்துக்காட்டை இயக்க,

```
ruby -e 'print "Hello Ruby!\n"'
Hello Ruby!
```

e-flag ஆனது ஒரு வரி நிரல்களை மட்டுமே இயக்க அனுமதிக்கும். ஆனால் பல *'-e'*-யை கொண்டு பல வரிகளை ஒரு *command line*-லேயே இயக்கலாம்.

3.2Interactive வாக ரூபியை இயக்குதல்:

ரூபி ஒரு *interpreted language*. அப்படியென்றால் ரூபி நிரலானது நிகழ்நேரத்தில் *compile* செய்யப்பட்டு இயக்கப்படும் என முந்தைய அத்தியாயங்களில் அறிந்தோம். *Interpreted language* ஆக இருப்பதில் உள்ள ஒரு சிறப்பு என்னவென்றால், ரூபி நிரலை *interpreter* ரில் நேரடியாக எழுதி நிகழ்நேரத்தில் இயக்க முடியும். இது ரூபி கற்றுக்கொள்ள, ஒரு சிறந்த வழியாகும். *IRB*

(Interactive Ruby Shell) என்ற மென்பொருள் இதற்கு பயன்படும்.

Windows-ல் *one-click installer*-ரை கொண்டு ரூபியை நிறுவியிருந்தால், *irb* யும் நிறுவப்பட்டிருக்கும். *Linux*-ல் *irb* நிறுவப்பட்டுள்ளதா என்பதை பின்வருமாறு அறிந்து கொள்ளலாம்.

```
irb -v
irb 0.9(02/07/03)
```

அதற்கான *version* விவரத்தை பெற இல்லையெனில் *irb*-யை நிறுவச்செய்வது அவசியம். *Red Hat or Fedora linux*-ல் பின்வருமாறு நிறுவலாம்.

```
su
yum install irb
```

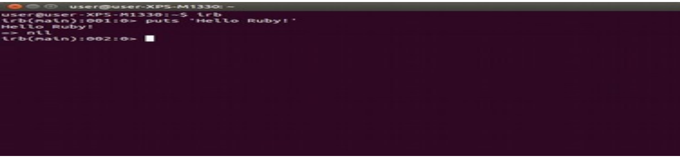
Debian, Ubuntu அல்லது மற்ற *debian derived linux distributions*-ல் *apt-get tool*-லை கொண்டு நிறுவ வேண்டும்.

```
sudo apt-get install irb
```

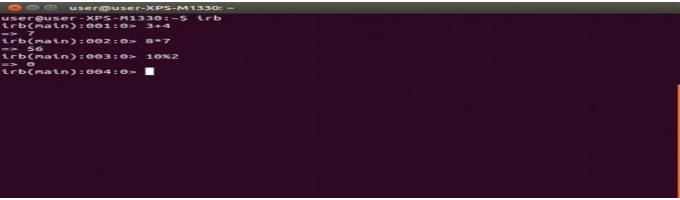
irb நிறுவப்பட்டவுடன் பின்வருமாறு *launch* செய்யவும்.

```
$ irb
irb(main):001:0>
```

இப்போது ரூபி நிரலை இயக்கத் தொடங்கலாம்.



இதில் கணக்குகள் (calculation) செய்ய முடியும்,



Irb prompt-ல் எதை எழுதினாலும் enter key -யை அழுத்தியவுடனே இயங்குகிறது.

3.3 ரூபியை file-லிருந்து execute செய்தல்:

Command line-ல் சில வரி ரூபி நிரலையே இயக்க முடியும். இதற்கு பொதுவான அணுகுமுறை, ரூபி நிரலை கோப்பில் (file-ல்) சேமித்து, அந்த கோப்பினை ரூபி interpreter-ல் இயக்க வேண்டும். இதை செய்ய, hello.rb என்று கோப்பினை உருவாக்கி, நீங்கள் விரும்பிய editor-ல் பின்வரும் நிரலைத் தட்டச்சு செய்யவும்.

```
print "Hello Ruby!\n"
print "Goodbye Ruby!\n"
```

இந்த நிரலை இயக்க command line-ல் பின்வருமாறு கொடுக்கவும்.

```
ruby hello.rb
Hello Ruby!
Goodbye Ruby!
```

3.4GNU/Linux ல் self contained ரூபி executable-லை உருவாக்குதல்:

command line-ல் பல்வேறு -e options-ஐப்பயன்படுதுவதை விட, ரூபி நிரலை கோப்பினில் சேமித்து இயக்குவது

மிகவும் எளிமையானது. எனினும்,மேலும் ஒரு படி முன்னே சென்று, ரூபி நிரல் உள்ள ரூபி கோப்பினை *ruby* என்கிற *prefix* இல்லாமல் இயக்க முடியும்.

இதற்கு *GNU/linux* -ல் ஒரு சிறப்பு வரியினை கோப்பின் முதல் வரியாக கொடுக்க வேண்டும். இது ரூபி நிரலினை இயக்கவல்ல ரூபி *interpreter* எங்கு உள்ளது என்பதை *environment* க்கு தெரிவிக்கும். இந்த சிறப்பு வரியானது '#', '!' மற்றும் ரூபி *executable path*-யும் கொண்டது. இதை "*Shebang*" எண்று சொல்வர்.

முதலாதவதாக, கணினியில் ரூபி எங்கே உள்ளது என்று தெரிந்து கொள்ள வேண்டும். இதை '*which*' *command*-டை கொண்டு கண்டுபிடிக்கலாம்.

```
which ruby
/usr/bin/ruby
```

மேலே உள்ள எடுத்துக்காட்டில் ரூபி */usr/bin/* -னில் உள்ளது. அதனால் நமது நிரலில் பின்வருமாறு மாற்றி எழுதலாம்.

```
#!/usr/bin/ruby
print "Hello Ruby!\n"
print "Goodbye Ruby!\n"
```

hello.rb script-டை இயக்கினால்,

```
./hello.rb
-bash: ./hello.rb: Permission denied
```

மேலே காணும் *output* வந்தால், *script*-டை இயக்க போதுமான அனுமதி *(permission)* இல்லை என்று அர்த்தம். இயங்குவதற்கான அனுமதியை *chmod* கொண்டு இந்த *script*-க்கு வழங்கலாம்.

```
chmod 755 hello.rb
```

இப்பொழுது இயக்கினால்,

```
./hello.rb
Hello Ruby!
Goodbye Ruby!
```

3.5Windows-ல் ரூபி file-லை Associate செய்தல்:

முந்தைய அத்தியாயத்தில் பார்த்த முறை *(shebang approach) windows*-ல் வேலை செய்யாது. *'.rb' file extension*-னை *window system*-வுடன் *configure* செய்ய *window file type association*-னை பயன்படுத்த வேண்டும்.

எடுத்துக்காட்டிற்கு *windows*-ல், *.doc* கோப்பினை *double click* செய்தால் அது தானாக *Microsoft word*-ல் திறக்கும் *(or) Microsoft word*-ஆல் திறக்கப்படும். ஏனென்றால் *.doc* கோப்புகளுக்கும் *word*-க்குமான இணைப்பு *(association)* கணினியில் கட்டமைக்கபட்டிருக்கும் *(configuration)*. அதேப்போல, *.rb* கோப்பினை ரூபியுடன் இணைக்க வேண்டும்.

Windows-ல் *ruby one-click installer*-ரை கொண்டு நிறுவியிருந்தால், *.rb files* தானாகவே ரூபியுடன் இணைக்கப்பட்டிருக்கும். ஆதலால் வேறொன்றும் தனியாக செய்ய வேண்டியதில்லை . *Command Prompt*-ல் *hello.rb* என்று *type* செய்தால் போதும், நமது எடுத்துக்காட்டு இயங்கும்.

இதே *one-click installer* இல்லாது மூலநிரலிலிருந்து ரூபியை நிறுவியிருந்தால் *.rb file*-லை ரூபியுடன் இணைக்க வேண்டும்.

இணைப்பு ஏற்கனவே கட்டமைக்கபட்டுள்ளதா என்று பார்க்க வேண்டும்.

```
C:\MyRuby>assoc .rb
File association not found for
```

```
extension .rb
```

இணைப்பு கட்டமைக்கப்படவில்லையென்றால், பின்வருமாறு செய்ய வேண்டும்.

```
C:\MyRuby>assoc .rb=rbFile
```

Rbfile type ஏற்கனவே உள்ளதா என்பதை சரிபார்க்கலாம்,

```
C:\MyRuby>ftype rbfile
File type 'rbfile' not found or no open
command associated with it.
```

ஏற்கனவே இல்லையெனில்,

```
C:\MyRuby>ftype
rbfile="D:\Ruby\bin\ruby.exe" "%1" %*
```

இந்த அமைப்பினை (Setting) பின்வருமாறு சரிபார்க்கலாம்.

```
C:\MyRuby>ftype rbfile
rbfile="D:\ruby\bin\ruby.exe" "%1" %*
```

PATHEXT environment variable-லில் .rb-யை

பின்வருமாறு சேர்க்க வேண்டும்.

```
C:\MyRuby>set PATHEXT=.rb;%PATHEXT%
```

அமைப்புகளையெல்லாம் கட்டமைத்தபின், *Command Prompt*-ல் கோப்பின் பெயரைத்தட்டச்சு செய்து *program*-யை இயக்கலாம். *.rb file* நீட்டிப்பு *(extension)* தேவையில்லை.

```
C:\MyRuby> hello
Hello Ruby
```

மேலே உள்ள படிகளை *(steps) Autoexec.bat*-ல் வைக்க வேண்டும். இல்லையெனில் உங்கள் *system reboot* செய்யும் ஒவ்வொரு முறையும் இணைப்பினை கட்டமைக்க செய்ய வேண்டும்.

4 ரூபி code-ல் comment செய்தல்:

Comment என்பது programmer-இன் பயன்பாட்டிற்காக நிரலில் எழுதப்படும் வரிகளாகும். நிரலிலுள்ள comment-களை interpreter இயக்க முயற்சிக்காது, நிராகரித்துவிடும். Comment ஒருவரியிலோ, பலவரிகளிலோ இருக்கலாம். மற்ற programmer-களால் பயன்படுத்தப்படும் library-கள் எழுதும் பொழுது, documentation-காக comment-கள் பயன்படுத்தப்படும். ரூபி documentation-க்கு பயன்படுத்தபடும் rdoc, இதற்கு ஒரு சிறந்த உதாரணம்.

நிரல் வரிகளை, comment செய்வதின் மூலம், interpreter-ஆல் இயக்கமுடியாமல் தடுக்கலாம். இது தற்காலிகமாக இருக்க வேண்டும். எழுதப்பட்ட நிரல், எதிர்பார்த்தபடி இயங்குகிறதா என்பதை சோதித்தபின், இது போன்ற தேவையற்ற, comment செய்யப்பட்ட நிரல் வரிகள் நீக்கப்படவேண்டும். இது ஒரு சிறந்த பழக்கமேயன்றி (best practice) கட்டாயமானதல்ல.

4.1 ஒரு வரியில் ரூபி comments:

ரூபியில் ஒரு வரி comment-யை '#' குறியீட்டை கொண்டு வரையறுக்கலாம். எடுத்துக்காட்டாக, ஒரு எளிய நிரலில் ஒரு வரி comments-டை சேர்க்கலாம்.

```
# This is a comment line - it explains
that the next line of code displays a
welcome message
print "Welcome to Ruby!"
```

பல வரிகளுக்கு பின்வருமாறு comments கொடுக்கலாம்.

```
# This is a comment line
# it explains that the next line of code
displays
# a welcome message
```

4.2Code உள்ள வரியில் comments-யை கொடுத்தல்:

ஒரேவரியில் நிரலைதொடர்ந்து, comments கொடுப்பது ஒரு பொதுவான பயிற்சியாகும். எடுத்தக்காட்டாக print statement —உள்ள வரியிலேயே comments-யை '#' குறியீட்டை தொடர்ந்து கொடுக்க வேண்டும்.

```
print "Welcome to Ruby!"     # prints the
welcome message
```

வரியில் '#' தொடர்ந்து வரும் எல்லாமே ruby interpreter-ல் நிராகரிக்கப்படும். '#' குறியீட்டை தொடர்ந்து வேறு நிரல் எழுதி அது இயங்கவேண்டும் என்று எதிர்ப்பார்க்க கூடாது. கூடுதலான நிரலை அடுத்த வரியில் தான் எழுத வேண்டும்.

4.3 பல வரிகளில் ரூபி comments:

ரூபியில் பலவரி comment-களை, =begin மற்றும் =end என்கிற குறியீடுகளை கொண்டு வரையறுக்கலாம். இவை 'comment block markers' என அறியப்படும்.

எடுத்துக்காட்டாக,

```
=begin
This is a comment line
it explains that the next line of code
displays
a welcome message
=end
```

5 ரூபியின் variables-யை புரிந்து கொள்ளல்:

Variable என்பது value ஐப் பயன்படுத்த உதவும் ஒரு வழியாகும் மற்றும் value-விற்கு பெயர் assign செயவதாகும். Variables integer முதல் string வரை பல்வேறு எல்லையிலுள்ள value-வை எடுக்கும். இந்த அத்தியாயத்தில் variables எப்படி declare மற்றும் மாற்றச் செய்வதைப் பார்க்கலாம்.

5.1 ரூபியின் constants:

ரூபி constant ஆனது ரூபி program execution முழுவதும், அதன் மதிப்பை மாற்றாமல் வைக்க பயன்படுவதாகும். Constants declaration-ல் variable-லின் பெயரின் தொடக்கம் capital letter-ல் இருக்க வேண்டும்.ஒரு பொதுவான விசயம் (convention) என்னவென்றால் constants-ன் variable பெயர் முழுவதும் uppercase-ல் இருப்பதாகும்.

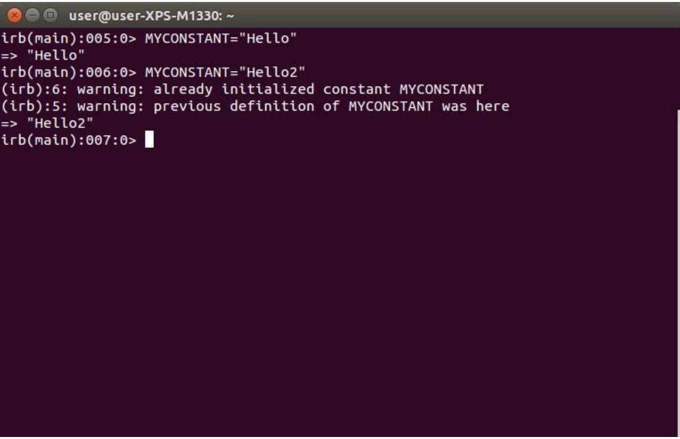
உதாரணத்துக்கு,

```
MYCONSTANT = "hello"
=> "hello"
```

மற்ற programming languages போல் இல்லாது, ரூபியில் constants-க்கு ஒதுக்கிய value-வை பின்னர் மாற்ற முடியும்.

அவ்வாறு செய்யும்போது ரூபி *interpreter* ஒரு எச்சரிக்கை *(warning)* கொடுக்கும், இருந்தபோதும் மாற்றத்தை அனுமதிக்கும்.

```
MYCONSTANT = "hello2"
```




```
(irb):34: warning: already initialized
constant Myconstant
=> "hello2"
```

Java மற்றும் *C* போன்ற மொழிகள் *strong* அல்லது *static variable*

typing பயன்படுத்தும். அப்படியென்றால், *variable declare* செய்யும்போது அது என்ன வகை *(type) variable* என்பதையும் அறிவிக்கவேண்டும். உதாரணமாக, ஒரு *variable*-லில் *integer*-ன் மதிப்பை வைக்க, அந்த *variable*-லை அறிவிக்கும் *(declare)* பொழுது, அதன் பெயருக்கு முன்பாக, *'Integer'* என குறிப்பிட வேண்டும். இதுப்போல் அறிவிக்கப்பட்டபின், அந்த *variable*-ன் வகையை மாற்ற இயலாது.

ஆனால் ரூபி ஒரு *dynamically typed language*. அதாவது, *variable*-லை உருவாக்கும்போது அதன் வகையை குறிப்பிடத்தேவையில்லை . ரூபியின் *interpreter* ஆனது *variable*-க்கு கொடுக்கும் மதிப்பைக்கொண்டு அதன் வகையை அறிந்துகொள்ளும். மற்றொரு நன்மை என்னவென்றால் ஒருமுறை *variable*-லை அறிவித்து, ஒரு மதிப்பைக்கொடுத்தபின் மற்றொரு வகையின் மதிப்பிற்கு *dynamic*-ஆக மாற்றிக்கொள்ளலாம்.

5.2Variable-லை declare செய்தல்:

Variable-க்கு மதிப்பைக்கொடுக்க, அதன் பெயர் மற்றும் மதிப்பை, *assignment operator(=)*க்கு இரு பக்கமும் தர வேண்டும். உதாரணத்துக்கு, *"Y"* என்ற *variable*-க்கு *10* என்கிற மதிப்பைக்கொடுக்க பின்வருமாறு எழுதலாம்.

```
Y = 10
```

பொதுவாக, வேறு scripting languages-ல் உள்ளது போல, ரூபியும் இணையான (parallel) assignment-யை ஆதரிக்கும். பல variables-க்கு மதிப்பை அளிக்க இது பயன்படும். அதை வழக்கமான முறையில், பின்வருமாறு செய்யலாம்.

```
a = 10
b = 20
c = 30
d = 40
```

மற்றொரு முறையாக, இணையான assignment-யை பின்வருமாறு செய்யலாம்.

```
a, b, c, d = 10, 20, 30, 40
```

5.3 ரூபியின் variable type-யை கண்டறிதல்:

ரூபி variable அறிவித்த பின், object class-ல் உள்ள kind_of? என்கிற method, அதன் வகையை கண்டுபிடிக்க உதவியாக இருக்கும். உதாரணத்துக்கு, நமது variable integer-ஆ என்பதை kind_of? Method-டை பயன்படுத்தி கண்டுப்பிடிக்கலாம்.

```
y.kind_of? Integer
=> true
```

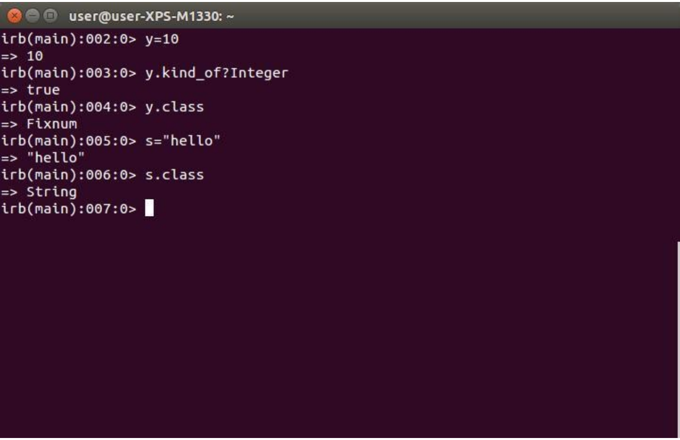
class method-டை பயன்படுத்தி நமது *variable* சரியாக எந்த *class*-யை சேர்ந்தது என்பதை கண்டுப்பிடிக்கலாம்.

```
y.class
=> Fixnum
```

இந்த *variable fixed number class*-யை சேர்ந்தது என்று அறியப்படுகிறது.

இதே வேலையை *string variable*-ம் செய்யலாம்.

```
s = "hello"
s.class
=> String
```



5.4Variable-லின் type-யை மாற்றுதல்:

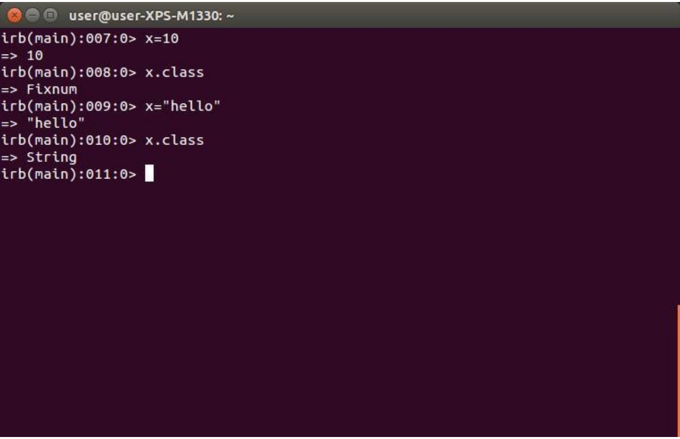
Variable வகையை மாற்றுவதற்கு எளிமையான வழி variable-க்கு புது மதிப்பை பொருத்துவதுதான். ரூபி, variable-லின் வகையை, புதிதாக கொடுக்கப்பட்ட மதிப்பைப் பொறுத்து மாற்றிக் கொள்ளும். உதாரணத்துக்கு, Integer value கொண்ட variable-லின் வகையை சரிர்க்கலாம்.

```
x = 10
```

```
=> 10
x.class
=> Fixnum
```

ஒரு வேளை, *x* என்கிற *variable*-லை *string*-ஆக மாற்ற வேண்டுமெனில், ஏதேனுமொரு *string*-யை *variable*-க்கு பொருத்தினால் போதும். ரூபி நமக்காக *variable* வகையை மாற்றி விடும்.

```
x = "hello"
=> "hello"
x.class
=> String
```



5.5Variable-லின் மதிப்பை மாற்றுதல்:

ஒரு variable-லிருந்து மதிப்பை பெற்று அதை வேறொரு variable type ஆக மாற்ற வேண்டுமெனில்,ரூபி variable classes-ல் methods உள்ளது. உதாரணத்துக்கு, fixnum class-ல் உள்ள to_f என்கிற method-டை பயன்படுத்தி integer மதிப்பை float-ஆக பெறலாம்.

```
y = 20
=> 20
y.to_f
=> 20.0
```

அதே போல, *to_s* என்கிற *method*-டை பயன்படுத்தி ரூபியில் *integer*-ரை *string*-ஆகப்பெற இயலும். இவ்வகை *to_** *method*-கள் வேண்டப்பட்ட வகையில், ஒரு புதிய மதிப்பைத்தருமேயன்றி, *variable*-ன் மதிப்பை மாற்றாது. *to_f*, *to_s* *method*-களைப்பயன்படுத்தியபிறகும், *y*-இன் மதிப்பு, *20*-ஆகவே இருக்கும்.

to_s *method*-ல் மாற்ற வேண்டிய *number base*-யை *argument*-ஆக கொடுக்க வேண்டும். *Number base* கொடுக்கவில்லையெனில் அதை *decimal*-ஆக எடுத்துக் கொள்ளும்.

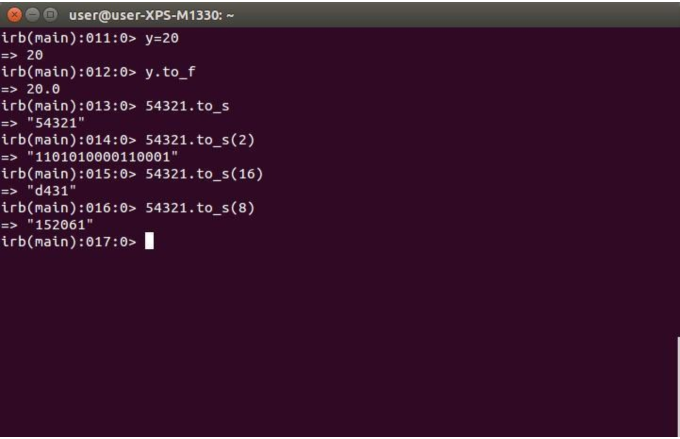
```
54321.to_s
=> "54321"
```

மாறுதலாக, *argument*-ல் *number base*-யை *2* என்று கொடுத்து *binary*-ஆக மாற்றலாம்.

```
54321.to_s 2
=> "1101010000110001"
```

Hexadecimal மற்றும் *octal*- ஆக மாற்ற:

```
54321.to_s 16
=> "d431"
54321.to_s 8
```




```
=> "152061"
```

to_s method-ல் number base-யை 1 முதல் 36 வரை பயன்படுத்தலாம்.

6 ரூபி variable scope:

Scope என்பது *program*-ல் *variable*-களின் எல்லைகளை வரையறுக்கும். ரூபியில் *variable scope* நான்கு வகைப்படும், அவை *local,global,instance* மற்றும் *class*. கூடுதலாக ரூபியில் *constant type*-ம் உண்டு. ஒரு *variable*-ன் பெயரின்முன்வரும் சிறப்பு குறியீட்டைப்பொருத்து அதன் எல்லை அறியப்படுகிறது.

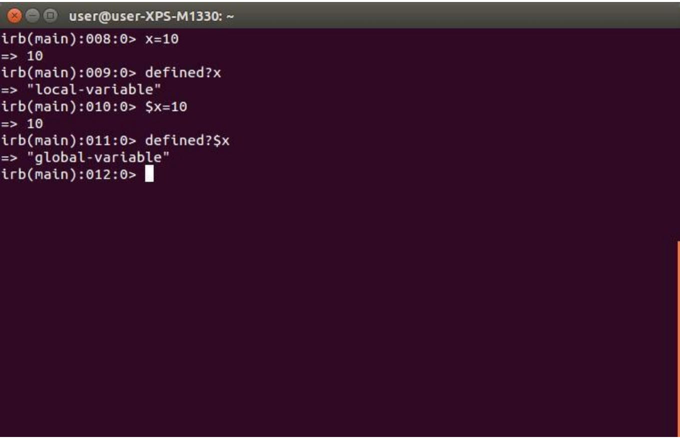
Name Begins With	*Variable Scope*
$	*A global variable*
@	*An instance variable*
[a-z] or _	*A local variable*
[A-Z]	*A constant*
@@	*A class variable*

கூடுதலாக ரூபியில் இரண்டு போலியான(pseudo) variables உண்டு. இதற்கு மதிப்பினைக்கொடுக்க இயலாது. ஒன்று nil, வெளிப்படையாக மதிப்பு அளிக்கபடாத variables-க்கு பொருத்தப்படும், மற்றொன்று self, தற்சமயம் பயன்பாட்டிலுள்ள object-டை குறிக்கும்.

6.1 ரூபி variable-லின் scope-யை கண்டறிதல்:

ஒரு variable-லின் பெயரை வைத்தே அதனின் scope-யை அறிந்து கொள்ளலாம். எனினும் நிரலில் scope-யை கண்டறிய defined? Method-டை பயன்படுத்தலாம். defined? Method கொடுக்கப்பட்ட variable-லின் scope-யை திருப்பியளிக்கும் அல்லது variable அறிவிக்கபடவில்லையெனில் 'nil'-யை திருப்பியளிக்கும்.

```
x = 10
=> 10
defined? x
=> "local-variable"
$x = 10
=> 10
defined? $x
=> "global-variable"
```



6.2 ரூபி local variable:

Local variable-ஆனது நிரலில் எந்த பகுதியில் அறிவிக்கப்படுள்ளதோ அந்த பகுதியில் மட்டுமே பயன்படுத்த முடியும். உதாரணத்துக்கு, ஒரு *local variable* ஆனது *method* அல்லது *loop*-ன் உள்ளே கொடுக்கப்பட்டிருந்தால் அதற்கு வெளியே அதை பயன்படுத்தமுடியாது. *Local variable*-லின் தொடக்கத்தில் *underscore*-ரோ அல்லது *lower case letter*-ரிலோ இருக்க வேண்டும் உதாரணத்துக்கு,

```
loopcounter = 10
_LoopCounter = 20
```

6.3 ரூபி global variables:

ரூபியின் global variable-லை ரூபி நிரலில் எங்கிருந்து வேண்டுமானாலும், எங்கு கொடுத்திருந்தாலும் பயன்படுத்த முடியும். Global variable-லின் பெயரை dollar sign-னை முதன்மையாக கொடுக்க வேண்டும். உதாரணத்துக்கு,

```
$welcome = "Welcome to Ruby Essentials"
```

Global variable பயன்படுத்துவதில் பிரச்சனை உள்ளது. நிரலில் எங்கிருந்து வேண்டுமானாலும் பயன்படுத்துவது மட்டுமில்லாது,அதை மாற்றவும் இயலும். இது பிழைகளை கண்டுப்பிடிப்பதை கடினமாக்கும்.

ரூபி இயக்க சூழல் (execution environment) பற்றிய விவரங்களைப்பெற சில முன் வரையறுக்கப்பட்ட (pre-defined) global variable-கள் உள்ளன. அதை பின்வரும்

அட்டவணையில் காணலாம்.

Variable Name	*Variable Value*
$@	*The location of latest error*
$_	*The string last read by gets*
$.	*The line number last read by interpreter*
$&	*The string last matched by regexp*
$ஸ்ரீ	*The last regexp match, as an array of subexpressions*
$n	*The nth subexpression in the last match (same as $ஸ்ரீ[n])*
$=	*The case-insensitivity flag*

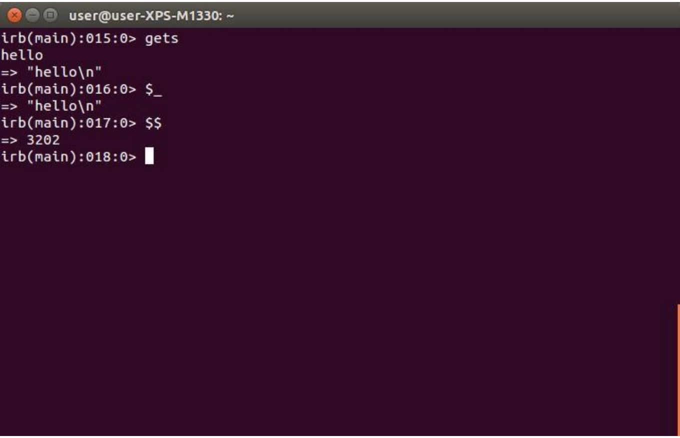
$/	*The input record separator*
$	*The output record separator*
$0	*The name of the ruby script file currently executing*
*$**	*The command line arguments used to invoke the script*
$$	*The Ruby interpreter's process ID*
$?	*The exit status of last executed child process*

உதாரணத்துக்கு, *gets method*-டை கொண்டு, தட்டச்சு இயந்திரத்திலிருந்து உள்ளீட்டைப்பெற்று, *$_ variable* கொண்டு கொடுக்கப்பட்ட *value*-வை பெற முடியும்.

```
irb(main):005:0> gets
hello
=> "hello\n"
irb(main):006:0> $_
```

```
=> "hello\n"
```

இதே போல, ரூபி *interpreter*-ன் *process ID*-யை கண்டுபிடிக்க முடியும்.




```
irb(main):007:0> $$
=> 17403
```

6.4 ரூபி class variables:

Class variable ஒரு *variable* அது *class*-ன் எல்லா *instances*-களாலும் பகிர்ந்து கொள்ளப்படும். அப்படியென்றால் ஒரே ஒரு *variable*-லின் மதிப்பானது அந்த *class*-ன் எல்லா *objects*-களாலும் பயன்படுத்தப்படும். மேலும் ஒரு *object instance variable*-லின் மதிப்பு மாற்றம் செய்தால் அது அந்த *class*-லுள்ள எல்லா *object instances*-களிலும் மாறும். *Java*-வின் *static variable*-க்கு இணையானதாக இதைக்கருதலாம்.

Class variable-லை அறிவிக்க, *variable* பெயரில் இரண்டு @ குறியீடுகள் (@@) முன்னதாக கொடுக்க வேண்டும். *Class variable* அறிவிக்கப்படும்பொழுதே அதற்கான மதிப்பு அளிக்கப்படவேண்டும். உதாரணத்துக்கு,

```
@@total = 0
```

6.5 ரூபி Instances variables:

Instance variables-ன் மதிப்பு ஒரு குறிப்பிட்ட *object instance*-க்கு மட்டும் சொந்தமானதாக இருக்கும். உதாரணத்திற்கு, ஒரு *class*-ல் *@total* என்கிற *instance variable* இருக்கிறதென வைத்துக் கொள்வோம். அந்த *@total*-லின் மதிப்பு ஒரு *object instance* மாற்றம் செய்தால்

அந்த *object*-டின் *@total* மதிப்பை மட்டுமே மாற்றும். அதே *class*-யை சேர்ந்த மற்ற *object*-களிலுள்ள *variable*-லின் மதிப்பை மாற்றாது.

Instance variable-ஐ அறிவிக்கும்பொழுது, *variable*-லின் பெயரில் முன்னதாக @ குறியீட்டை சேர்க்க வேண்டும்.

```
@total = 10
```

6.6 ரூபி Constant scope:

பொதுவாக *constant* -ற்கு கொடுத்த மதிப்பை மாற்ற கூடாது. ஆனால் ரூபியில் *constant*-ன் மதிப்பை மாற்ற இயலும். ரூபி *interpreter* ஒரு *warning message*-யை கொடுக்கும். இருப்பினும் *constant*-ன் மதிப்பை மாற்றிக் கொள்ளும்.

Class அல்லது *module*-லில் *constants*-யை கொடுத்தால், அது அந்த *class* அல்லது *module*-ன் முன்னொட்டுடன் *(prefix)* மட்டுமே பயன்படுத்தும்படியாக இருக்கும்.

```
module ConstantScope
  CONST_EXAMPLE = “This is a constant”
end
ConstantScope::CONST_EXAMPLE => “This is a
constant”
```

```
CONST_EXAMPLE=> uninitialized constant
CONST_EXAMPLE
```

Class அல்லது *module* வெளியில் கொடுத்தால் அது *global scope* ஆகும்.

7 ரூபி number classes மற்றும் conversions:

ரூபியில் எல்லாமே *object* தான். இதில் ஆச்சரியப்படும் விசயம் என்னவென்றால் ரூபியில் எண்கள் கூட *object* தான். பெரும்பாலான நிரலாக்க மொழிகள் எண்களை *primitives* ஆக கருதும். ஆனால் ரூபியில் எண்கள், எழுத்துக்கள் என எல்லாமே *class* தான். அவற்றுகான *methods* ஐ நாம் இயக்கிப் பார்க்கலாம். எல்லா எண் வகைகளுக்கும் அதற்கான *class* ரூபியில் உள்ளது. அதிலுள்ள *method*-களைக்கொண்டு எண்களை கையாளமுடியும்.

7.1 ரூபி number classes:

ரூபியில் உட்பொதிந்த *(builtin)* எண்களுக்கான *classes* உண்டு. அதில் பொதுவாக பயன்படுத்தும் *classes*-யை இந்த பகுதியில் காணலாம்.

Integer class:

எல்லா *class*-களுக்கும் இது அடிப்படையான *class* ஆகும். பின்வரும் *classes* எல்லாம் இதிலிருந்து தருவிக்கப்பட்டவை *(derived)*.

Fixnum class:

Fixnum-ன் அதிகபட்ச எல்லை ஆனது, code எந்த system-ல் execute செய்கிறோமோ அதனின் architecture பொறுத்தே அமையும். ஒருவேளை fixnum, system architecture எல்லையை தாண்டினால், அதன் value ஆனது bignum ஆக interpreter-னால் மாற்றப்படும்.

Bignum class:

Bignum objects ஆனது ரூபி fixnum class-லின் எல்லையை தாண்டிய integer மதிப்பை வைத்து கொள்ளும். Bignum object கணக்கீடு திருப்பி அனுப்பும் விடை ஆனது fixnum-ல் பொருந்தினால், விடை fixnum-ஆக மாற்றப்படும்.

Rational class:

விகிதமுறு எண் என்பது ஒரு எண்ணாகும், இது fraction(p/q)-ல் கொடுக்கப்படும். இதில் p-தொகுதி எண் மற்றும் q-வகுக்கும் எ என்பர். விகிதமுறு இல்லாத எண்ணையை விகிதமுறா எண்கள் என்பர்.

7.2 ரூபியில் numbers-யை மாற்றுதல்:

ரூபியில் integer மற்றும் float methods பயன்படுத்தி எண்களை ஒரு வகையிலிருந்து மற்றொரு வகையாக மாற்ற முடியும். மாற்ற வேண்டிய மதிப்பை argument-ஆக

இந்த *methods*-களுக்கு கொடுக்க வேண்டும்.

Floating Point Number-ஐ Integer-ஆக மாற்றுதல்:

```
Integer (10.898)
=> 10
```

String-ஐ Integer-ஆக மாற்றுதல்

```
Integer ("10898")
=> 10898
```

Hexadecimal Number-ஐ Integer-ஆக மாற்றுதல்

```
Integer (0xA4F5D)
=> 675677
```

Octal Number-ஐ Integer-ஆக மாற்றுதல்

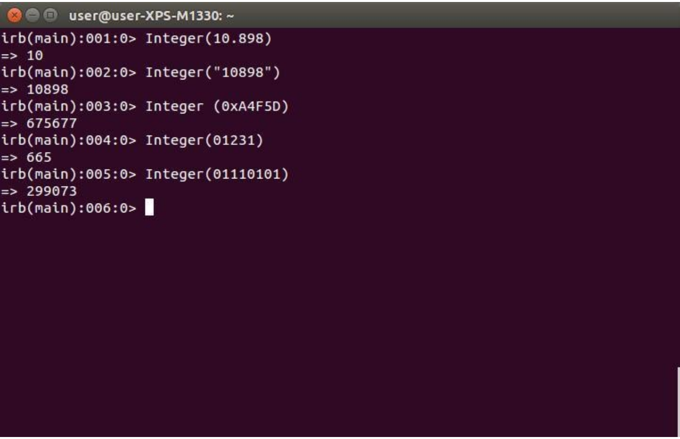
```
Integer (01231)
=> 665
```

Binary Number-ஐ Integer-ஆக மாற்றுதல்

```
Integer (01110101)
=> 299073
```

Character-ஐ ASCII Character Code--ஆக மாற்றுதல்

```
Integer (?e)
=> 101
```



அதேபோல் *float method* பயன்படுத்தி அதன் மதிப்பை *floating point* ஆக மாற்றலாம்.

Integer-ஐ Floating Point --ஆக மாற்றுதல்

```
Float (10)
```

```
=> 10.0
```

String-ஐ Floating Point--ஆக மாற்றுதல்

```
Float ("10.09889")
=> 10.09889
```

Hexadecimal Number-ஐ Floating Point--ஆக மாற்றுதல்

```
Float (0xA4F5D)
=> 675677.0
```

Octal Number-ஐ Floating Point--ஆக மாற்றுதல்

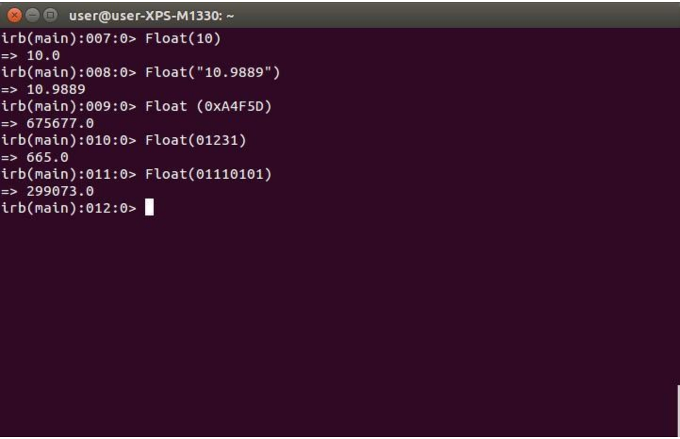
```
Float (01231)
=> 665.0
```

Binary Number-ஐ Floating Point--ஆக மாற்றுதல்

```
Float (01110101)
=> 299073.0
```

Character-ஐ Floating Point ASCII Character Code--ஆக மாற்றுதல்

```
Float (?e)
=> 101.0
```



8 ரூபி methods:

ரூபி methods ஆனது நிரலை ஒருங்கிணைக்கவும், மறுபயன்பாடு செய்யவும் வழி செய்கிறது. நீண்ட ரூபி நிரலாக எழுதுவதற்கு பதிலாக நிரலை தர்க்க ரீதியில் (logical group) ஒருங்கிணைத்து நமது நிரலில் எங்கு தேவையோ அங்கு மறுபயன்பாடு செய்து கொள்ளலாம். இதனால் ஒரே நிரலை மீண்டும் மீண்டும் எழுத வேண்டியதில்லை. Method-டை பயன்படுத்துவது மிகவும் எளிது. இதற்கு இரண்டே விசயம் செய்தால் போதும். ஒன்று method-டை உருவாக்குதல் மற்றொன்று அதை அழைத்தல் ஆகும்.

பிற மொழிகளில் உள்ள Function என்பதையே இங்கு Method என்கிறோம்.

8.1 ரூபி method-டை உருவாக்குதல் மற்றும் அழைத்தல்:

ரூபி method-ன் அமைப்பு (syntax) பின்வருமாறு.

```
def name( arg1, arg2, arg3, ... )
   .. ruby code ..
   return value
end
```

இதில் *name* ஆனது *method*-ன் பெயர் ஆகும். *Method*-ன் பெயரை பயன்படுத்தியே *method*-யை அழைக்க வேண்டும். *Arguments* ஆனது *method* செயல்படத்தேவையான மதிப்பை அனுப்புவதாகும். *Ruby code* என்பது *method*-ன் *body* ஆகும், செயல்பாடு இங்கே நடைபெறும். *Return statement* கட்டயமானதல்ல, இது நிரலில் *method*-டை அழைக்குமிடத்திற்கு மதிப்பை திருப்பி அனுப்பும்.

பின்வரும் எடுத்துக்காட்டானது *string display* செய்வதாகும். இதில் *method*-டை உருவாக்குதல் மற்றும் அழைத்தல் செய்வதை பார்க்கலாம்.

```
def saysomething
    puts "Hello"
end

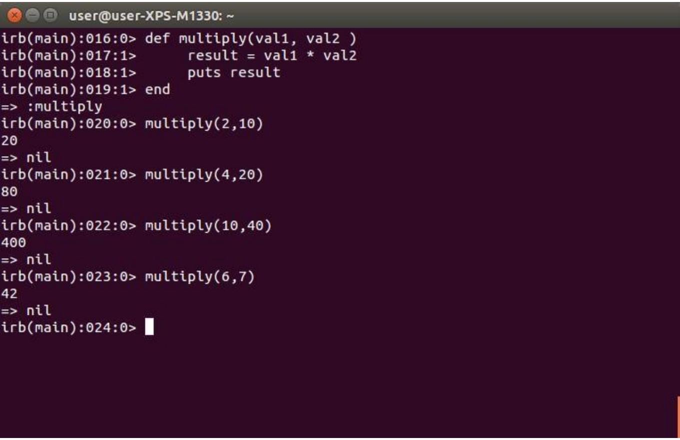
saysomething
```



8.2Method-க்கு arguments அனுப்புதல்:

மேலே உள்ள எடுத்துக்காட்டில் *method*-க்கு எந்த *arguments* அனுப்பப்படவில்லை. *Arguments* மதிப்பை கொண்டு *method*-ல் கணக்கீடு செய்வதை பின்வரும் எடுத்துக்காட்டில் பார்க்கலாம்.

```
def multiply(val1, val2 )
     val1 * val2
end
multiply 2, 10   # 20
multiply 4, 20   # 80
multiply 10, 40   # 400
multiply 6, 7   #42
```



இந்த எடுத்துக்காட்டில், *method* பலமுறை அழைக்கப்பட்டுள்ளது. இதில் *arguments*-யை அனுப்ப, அதை *method*-ல் கணக்கீடல் செய்து விடையைத் தருகிறது. இதை *method* இல்லாமல் செய்ய வேண்டுமானால் இந்த நிரலை 4 முறை எழுத வேண்டியதிருக்கும். ஆதலால் நிரலை *method*-ல் வைப்பதானால் நிரல் மறுபயன்பாடு செய்யப்படுகிறது. இது ஒரு சிறந்த வழியாகும்.

8.3Method-க்கு பல்வேறு arguments-யை அனுப்புதல்:

*முந்தைய பகுதில், ஒரு குறிப்பிட்ட, நிலையான எண்ணிக்கை கொண்ட arguments-யே method எடுத்துக்கொண்டது. சிலநேரங்களில் எவ்வளவு arguments தேவை என்பது நமக்கே தெரியாது. இதை *args-யை கொண்டு செய்யலாம். Method உருவாக்கும்போது இதை கொடுக்க வேண்டும். Method-ல் arguments-ஆக அனுப்பப்படும் மதிப்பை array-ல் வைத்து method-ல் செயல்படுத்தலாம்.*

```
def displaystrings( *args )
        args.each {|string| puts string}
end

displaystrings("Red")
Red

displaystrings("Red", "Green")
Red
Green

displaystrings("Red", "Green", "Blue")
Red
Green
Blue
```

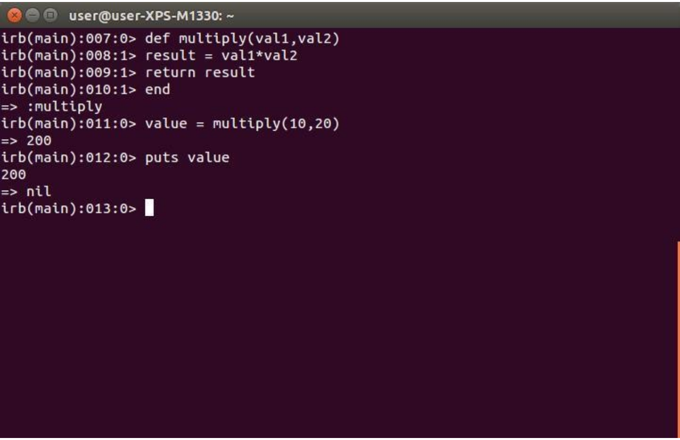
```
user@user-XPS-M1330: ~
irb(main):001:0> def displaystrings(*args)
irb(main):002:1> args.each {|string| puts string}
irb(main):003:1> end
=> :displaystrings
irb(main):004:0> displaystrings("Red")
Red
=> ["Red"]
irb(main):005:0> displaystrings("Red","Green")
Red
Green
=> ["Red", "Green"]
irb(main):006:0> displaystrings("Red","Green","Blue")
Red
Green
Blue
=> ["Red", "Green", "Blue"]
irb(main):007:0>
```

8.4Function-லிருந்து விடையை திருப்பி அனுப்புதல்:

Return statement-டை பயன்படுத்தி method-லிருந்து மதிப்பை திருப்பி அனுப்பமுடியும். ஒரு method திருப்பியனுப்பும் மதிப்பை assignment(=) operator=ரை கொண்டு பெற வேண்டும். பின்வரும் எடுத்துக்காட்டில், method உருவாக்கி arguments-லிலுள்ள மதிப்பை பெருக்கல் செய்து விடையை திருப்பி அனுப்புகிறது.

.

```
def multiply(val1, val2 )
     result = val1 * val2
     return result
end
```




```
value = multiply( 10, 20 )
puts value
```

மேலே உள்ள எடுத்துக்காட்டில் 10 மற்றும் 20, arguments ஆக multiply method-க்கு அனுப்பபடுகிறது. Method ஆனது மதிப்பை பெருக்கல் செய்து அதன் விடையை திருப்பி

அனுப்புகிறது. திருப்பி அனுப்படும் விடையை ஒரு *variable*-லில் வைக்கப்படுகிறது. அப்புறம் அதை *puts*-யை பயன்படுத்தி காட்சிப்படுத்தப்படுகிறது.

இதில் கவனிக்க வேண்டிய விசயம், *method* ஒரு மதிப்பை அல்லது *object*-டைதான் திருப்பி அனுப்பும். நிறைய மதிப்பை திருப்பி அனுப்ப வேண்டுமெனில் ஒரு *array* வைத்து அனுப்ப வேண்டும்.

ரூபியில், ஒரு *method*-லிருந்து வெளிப்படையாக, ஒரு மதிப்பை திருப்பியனுப்பவேண்டிய அவசியமில்லை. *method*-ன் கடைசிவரி இயக்கத்தில் கிடைக்கும் மதிப்பு எப்பொழுதும் திருப்பியனுப்பப்படும். மேற்கண்ட எடுத்துக்காட்டை பின்வருமாறு மாற்றியெழுதலாம்

```
def multiply(val1, val2 )
     val1 * val2
end

multiply 10, 20 # 200
```

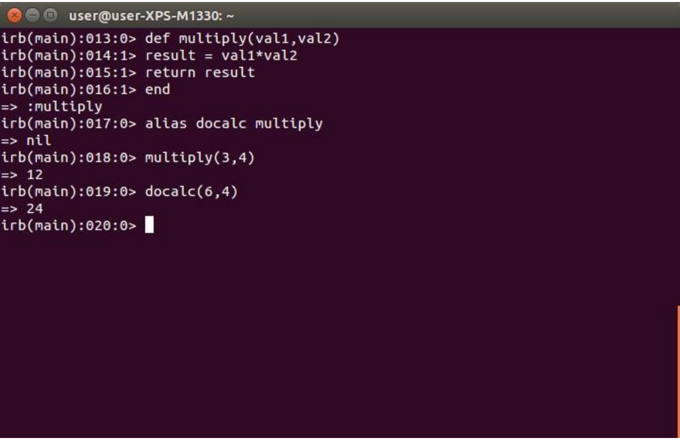
```
2.2.0 :001 > def multiply(val1, val2)
2.2.0 :002?>   val1 * val2
2.2.0 :003?>   end
 => :multiply
2.2.0 :004 > multiply 10, 20
 => 200
2.2.0 :005 >
```

8.5 ரூபி method-க்கு வேறுபெயர்(aliases) வைத்தல்:

ரூபி, method-க்கு வேறுபெயர் வைக்க அனுமதிக்கும். அவ்வாறு வேறுபெயர் வைக்கும் பொழுது, ரூபி, அந்த method-க்கு ஒரு நகலை உருவாக்கி அதற்கு இந்த புதிய பெயரை வைத்துவிடும். (எந்த பெயரையும் கொண்டு அந்த method-டை அழைக்க முடியும்) உதாரணத்திற்கு,

```
def multiply(val1, val2 )
     result = val1 * val2
     return result
end

alias docalc multiply
```




```
docalc( 10, 20 ) # 200

multiply( 10, 20 ) # 200
```

இவ்வாறு நகல்களை உருவாக்கி பயன்படுத்துதல், ரூபியின் பொதுவான பயன்பாடாகும். ஏற்கனவே வரையறுக்கப்பட்ட method-இல் மாற்றங்கள் செய்தால், அது இந்த புதிய நகலில் பிரதிபலிக்காது.

```
def multiply(val1, val2 )
     val1 * val2
end

alias docalc multiply

docalc 10, 20 # 200
multiply 10, 20 # 200

def multiply(val1, val2)
    p “This method adds two numbers”
    val1 * val2
end

docalc 10, 20 # 200
multiply 10, 20
=> “This method adds two numbers”
=> 30
```

```
2.2.0 :033 > def multiply(val1, val2)
2.2.0 :034?>   val1 * val2
2.2.0 :035?>   end
 => :multiply
2.2.0 :036 > alias docalc multiply
 => nil
2.2.0 :037 > docalc 10, 20
 => 200
2.2.0 :038 > multiply 10, 20
 => 200
2.2.0 :039 > def multiply(val1, val2)
2.2.0 :040?>   p "This method adds two numbers"
2.2.0 :041?>   val1 + val2
2.2.0 :042?>   end
 => :multiply
2.2.0 :043 > docalc 10, 20
 => 200
2.2.0 :044 > multiply 10, 20
"This method adds two numbers"
 => 30
2.2.0 :045 >
```

மேற்கண்ட உதாரணத்தில், *multiply method*-ன் வரையறையை மாற்றிய பிறகும், *docalc method* பழைய வரையறையின்படி இயங்குவதைக்காணலாம்.

மூன்றாம் தரப்பு *(third party) library*-களை பயன்படுத்தும்பொழுது, அவற்றிலுள்ள *method*-ன் வரையறையை நம்தேவைகேற்ப, கூடுதல் செயல்பாட்டை உட்புகுத்த இந்த முறையை பயன்படுத்தலாம். *multiply*

method ஒரு மூன்றாம் தரப்பு *library*-யிலிருக்கிறது. அதன் வரையறை நமக்கு தெரியவில்லையென வைத்துக்கொள்வோம். இந்த *method* என்ன செய்கிறது என ஒரு *print statment*-ஐ இதனுடன் இணைக்கவேண்டும் என வைத்துக்கொள்வோம். *multiply method*-க்கு ஒரு நகலை உருவாக்கி, *multiply*-யை மறுவரையறை செய்வதன்மூலம், நமக்குவேண்டிய மாற்றங்களை செய்துகொள்ளலாம்.

```
def multiply(val1, val2 )
     val1 * val2
end

alias docalc multiply

docalc 10, 20 # 200
multiply 10, 20 # 200

def multiply(val1, val2)
    p “This method multiplies two numbers”
    docalc val1, val2
end

docalc 10, 20 # 200
multiply 10, 20
=> “This method multiplies two numbers”
```

```
2.2.0 :045 > def multiply(val1, val2)
2.2.0 :046?>   val1 * val2
2.2.0 :047?>   end
 => :multiply
2.2.0 :048 > alias docalc multiply
 => nil
2.2.0 :049 > docalc 10, 30
 => 300
2.2.0 :050 > multiply 10, 30
 => 300
2.2.0 :051 > def multiply(val1, val2)
2.2.0 :052?>   p "This method multiplies two numbers"
2.2.0 :053?>   docalc val1, val2
2.2.0 :054?>   end
 => :multiply
2.2.0 :055 > docalc 10, 30
 => 300
2.2.0 :056 > multiply 10, 30
"This method multiplies two numbers"
 => 300
2.2.0 :057 >
```

```
=> 200
```

9 ரூபியின் ranges:

ரூபி ranges-என்பது ஒரு தரவு தொகுப்பு (dataset), அதில் ஆரம்பம் முதல் கடைசி வரை உள்ள மதிப்பான ஒரு தருக்க தொடர்ச்சியுடன் (logical sequence) இருக்கும். Range-ல் உள்ள மதிப்புகள் எண்களாகவோ, குறியீடுகளாகவோ, string அல்லது object ஆகவோ இருக்கலாம்.

9.1 ரூபியின் sequence range:

ரூபியில் sequence ranges-யை பயன்படுத்தி அடுத்தடுத்த மதிப்புகளை உருவாக்கலாம். அவற்றுள் ஆரம்ப மதிப்பு, இறுதி மதிப்பு மற்றும் இடையிலுள்ள எல்லை மதிப்புகள் அடங்கும்.

இத்தகைய range உருவாக்க இரண்டு operators இருக்கிறது. ஒன்று இறுதி மதிப்பையும் உள்ளடக்கிய (inclusive) இரண்டு புள்ளிகள் கொண்ட operator (..) மற்றொன்று இறுதி மதிப்பை உள்ளடக்காத (exclusive) மூன்று புள்ளிகள் கொண்ட operator (...). Inclusive operator-ல் ஆரம்பம் மற்றும் இறுதி மதிப்பு வரிசையில் அடங்கும். Exclusive range operator இறுதி மதிப்பு வரிசையில் அடங்காது.

```
1..10    # Creates a range from 1 to 10
inclusive
1...10   # Creates a range from 1 to 9
```

range-களை array-ஆக மாற்ற ரூபியில் to_a method-டை பயன்படுத்த வேண்டும். உதாரணத்திற்கு,

```
(1..10).to_a
=> [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

(1...10).to_a
=> [1, 2, 3, 4, 5, 6, 7, 8, 9]
```

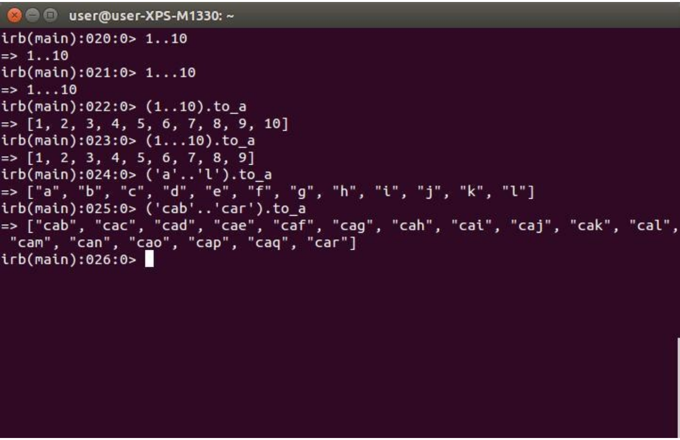
ஏற்கனவே சொன்னது போல, எல்லைகளின் மதிப்பை எண்கள் மட்டும் என்று கட்டுப்படுத்த முடியாது. குறியீடு சார்ந்த எல்லையையும் உருவாக்க முடியும்,

```
('a'..'l').to_a
=> ["a", "b", "c", "d", "e", "f", "g",
"h", "i", "j", "k", "l"]
```

வார்த்தை சார்ந்த எல்லையையும் பின்வருமாறு உருவாக்கலாம்.

```
('cab'..'car').to_a
=> ["cab", "cac", "cad", "cae", "caf",
```

```
"cag", "cah", "cai", "caj", "cak", "cal",
"cam",
"can", "cao", "cap", "caq", "car"]
```



எல்லையின் மதிப்புகள் objects-ஆகவும் இருக்கலாம். object-கள் கொண்ட range-ஐ உருவாக்க வேண்டுமெனில், அதிலுள்ள object-ஆனது, அதற்கடுத்த object-ஐ succ என்ற method மூலம் தரவல்லதாகவும், <=> operator கொண்டு ஒப்பிடக்கூடியதாகவும் இருக்கவேண்டும்.

ரூபியில் எல்லாமே object தான். அதேப்போல் range-ம்

Range என்ற *class*-ன் *object* தான். *Range class*-ல் பல *methods* உள்ளன,

```
words = 'cab'..'car'

words.min            # வரிசையிலுள்ள சிறிய மதிப்பை பெறுவதற்கு
=> "cab"

words.max            # வரிசையிலுள்ள பெரிய மதிப்பை பெறுவதற்கு
=> "car"

words.include?('can') # ஒரு மதிப்பு வரிசையில் உள்ளதா என அறிய
=> true

words.reject {|subrange| subrange < 'cal'}
# கொடுக்கப்பட்ட நிபந்தனைக்கு உட்படும் மதிப்புகளை நிராகரிக்க
=> ["cal", "cam", "can", "cao", "cap",
"caq", "car"]

words.each {|word| puts "Hello " + word} #
வரிசையிலுள்ள ஒவ்வொரு மதிப்பையும் கொண்டு ஒரு வேலையை செய்ய
Hello cab
Hello cac
Hello cad
```

Hello cae
Hello caf
Hello cag
Hello cah
Hello cai
Hello caj
Hello cak
Hello cal
Hello cam
Hello can
Hello cao
Hello cap
Hello caq
Hello car

```
user@user-XPS-M1330: ~
irb(main):030:0> words='cab'..'car'
=> "cab".."car"
irb(main):031:0> words.min
=> "cab"
irb(main):032:0> words.max
=> "car"
irb(main):033:0> words.include?('can')
=> true
irb(main):034:0> words.reject {|subrange| subrange < 'cal'}
=> ["cal", "cam", "can", "cao", "cap", "caq", "car"]
irb(main):035:0> words.each {|word| puts "Hello " + word}
Hello cab
Hello cac
Hello cad
Hello cae
Hello caf
Hello cag
Hello cah
Hello cai
Hello caj
Hello cak
Hello cal
Hello cam
Hello can
Hello cao
Hello cap
Hello caq
Hello car
=> "cab".."car"
irb(main):036:0>
```

9.2 ரூபி ranges as conditional expressions:

Conditional expressions-னில் ரூபி *ranges* பயன்படுத்தலாம். பின்வரும் எடுத்துக்காட்டில் கொடுக்கப்பட்ட எண் ஒன்றுக்கும் ஐந்துக்கும் இடையில் உள்ளதா என *ranges*-ஐ *conditional expression*-இல் பயன்படுத்தி அறியலாம்.

```
while a = gets.chomp.to_i
    puts “lies between 1 and 5” if((1..5)
=== a)
end
```

```
2.2.3 :014 > while a = gets.chomp.to_i
2.2.3 :015?>   puts "lies between 1 and 5 " if ((1..5) === a)
2.2.3 :016?>   end
1
lies between 1 and 5
2
lies between 1 and 5
3
lies between 1 and 5
4
lies between 1 and 5
5
lies between 1 and 5
6
7
78
9
90
```

கீழ்கண்ட எடுத்துக்காட்டில் 'start' என்ற உள்ளீட்டிற்கும், 'end' என்ற உள்ளீட்டிற்கும் இடையில் கொடுக்கப்படும் உள்ளீடுகள் மட்டும் திரையில் பதிக்கப்படும்.

```
while input = gets
```

```
 puts input + " triggered" if input =~
/start/ .. input =~ /end/
end
```

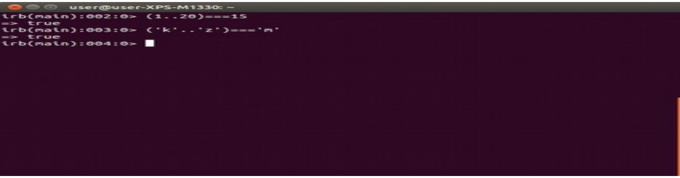


9.3 ரூபி எல்லை இடைவெளிகள்:

குறிப்பிட்ட எல்லைக்குள் ஒரு எண்ணோ அல்லது குறிப்பிட்ட எழுத்துக்கள் குழுவில் ஒரு எழுத்தோ இருக்கிறதா என்பதை கண்டுபிடிக்க(===) என்ற equality operator-ரை பயன்படுத்தலாம்.

```
(1..20) === 15
=> true
```

```
('k'..'z') === 'm'
=> true
```



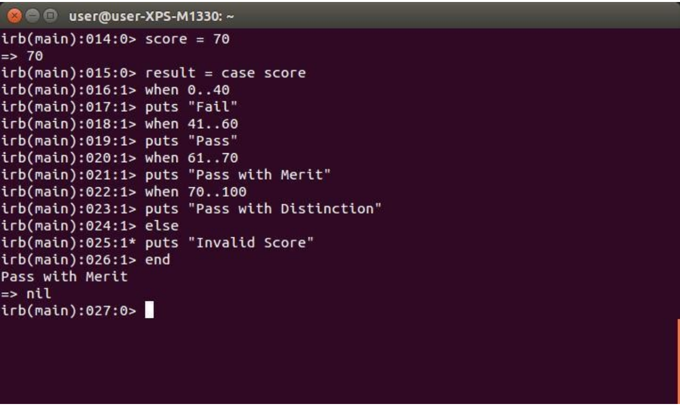
9.4Case statement-ல் ranges:

Ranges case statement-உடன் சேரும்போது மிகவும் சக்தி வாய்ந்தது ஆகிறது. பலவரிகள் வரை நீளக்கூடிய நிரல்களை, மிகசிலவரிகள் கொண்டு, கச்சிதமாக எழுத இது உதவுகிறது

```
score = 70

result = case score
   when 0..40
                puts  "Fail"
   when 41..60
                puts "Pass"
   when 61..70
```

```
                 puts  "Pass with Merit"
   when 71..100
                 puts "Pass with
Distinction"
   else "Invalid Score"
end
```




```
puts result
```

10 ரூபி array

ரூபி *variables* அத்தியாயத்தில் சொன்னதுபோல தரவுகளை நினைவக இடத்தில் வைப்பது மாறிகள் *(variables)* எனப்படும். பல்வேறு மாறிகளை ஒருகிணைத்து தன்னுள் கொண்டிருக்கும் பொருளாக *(object)* மாற்றுவது இன்றியமையாதாகும். இதை ரூபி *array*-யை கொண்டு செய்யலாம். இந்த அத்தியாயத்தில் *array*-யின் அறிமுகம், *array* உருவாக்குதல் மற்றும் கையாளுதலை காணலாம்.

10.1 ரூபி array என்றால் என்ன?:

ரூபியில் *array* ஒரு பொருளாகும். அதில் பல உருப்படிகள் *(items)* இருக்கும், அது எந்த வகையான மாறியாகவும் *(string, integer, fixnum, hash, objects, arrays etc)* இருக்கலாம். பல பரிமாணங்கள் *(multidimensional)* கொண்ட *array*-ஐ உருவாக்க, அதிலுள்ள ஒவ்வொரு உருப்படியும் ஒரு *array*-வாக இருக்கவேண்டும். இவ்வாறு பல உருப்படிகளை குழுவாக வைத்து ஒரு *array*-வை உருவாக்கியபின்பு, அவற்றை அகரவரிசைப்படியோ *(alphabetical order)*, அல்லது எண்வரிசைப்படியோ *(numerical order)* வரிசைப்படுத்துவது *(sorting)*,

உருப்படிக்கு கொடுக்கப்பட்ட மதிப்பை மாற்றுவது, நீக்குவது மற்றும் குழுவான உருப்படிகளை ரூபி செயற்கூற்றிற்கு *argument* ஆக அனுப்புவது போன்ற பலவற்றை செய்யலாம்

10.2 ரூபியில் array எப்படி உருவாக்குவது:

ரூபியில் *array* உருவாக்க பல்வேறு முறைகள் உள்ளன. ரூபி *array class*-யை பயன்படுத்தி *array* உருவாக்கலாம். காலியான *array* உருவாக்க *array*-யிலுள்ள *new* செயற்கூற்றை பயன்படுத்தி பின்வருமாறு செய்யலாம்.

```
days_of_week = Array.new
```

இதில் *days_of_week* என்கிற *array* காலியாக உள்ளது. *Array* காலியாக உள்ளதா என்பதை, *array class*-யிலுள்ள *empty?* செயற்கூற்றின் மூலம் சரிப்பார்க்கலாம். *Array* காலியாக இருந்தால் *array class true*-வை திருப்பி அனுப்பும்.

```
days_of_week.empty?
=> true
```

Array-வை துவக்க *(initialize)*, *array* அளவை *argument*-

ஆக *new* செயற்கூற்றிற்கு அனுப்ப வேண்டும்.

```
days_of_week = Array.new(7)
=> [nil, nil, nil, nil, nil, nil, nil]
```

Array-ல் உள்ள கூறுகள் எல்லாம் *nil* ஆக இருக்கும்.

10.3Array-ஐ விரிவுபடுத்துதல்:

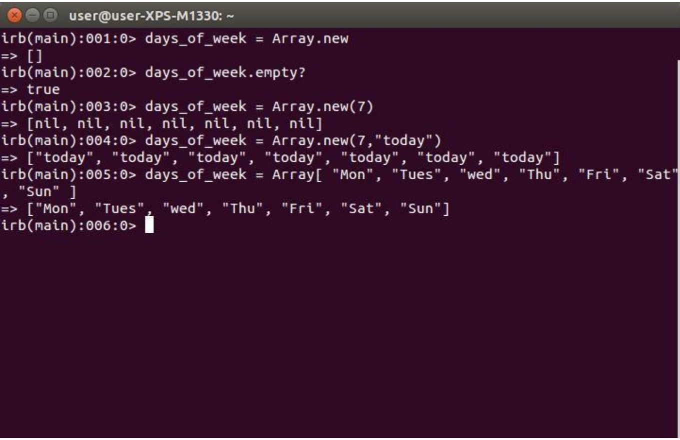
Array-யை உருவாக்கியப்பின் அதை விரிவுபடுத்தலாம். ஒரே மதிப்பை எல்லா கூறுகளுக்கும் கொடுக்க *array* உருவாக்கும்போதே *new* செயற்கூற்றிற்கு அதை அனுப்ப வேண்டும்.

```
days_of_week = Array.new(7, "today")
=> ["today", "today", "today", "today",
"today", "today", "today"]
```

மற்றொரு வழியாக, *array class*-யிலுள்ள*[] method*-டை பயன்படுத்தி கூறுகளுக்கு ஒன்றன்பின் ஒன்றாக மதிப்பு கொடுக்க வேண்டும்.

```
days_of_week = Array[ "Mon", "Tues",
"Wed", "Thu", "Fri", "Sat","Sun" ]
=> ["Mon", "Tues", "Wed", "Thu", "Fri",
```

```
"Sat", "Sun"]
```



Array பெயர் மற்றும் square bracket-ல் மதிப்பு மட்டும் கொடுத்தால் போதும்,

```
days_of_week = [“Mon”, “Tues”, “Wed”,
“Thu”, “Fri”, “Sat”, “Sun]
=> [“Mon”, “Tues”, “Wed”, “Thu”, “Fri”,
“Sat”, “Sun]
```

இது array உருவாக்குவதுடன் மதிப்பையும் சேர்க்கிறது.

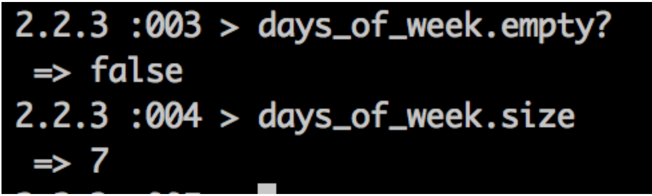
10.4 ரூபி array பற்றி விவரங்களை கண்டறிதல்:

Array உருவாக்கியப்பின்,array-யையும் அதன் கூறுகளையும் பற்றிய விவரங்களை பெறலாம். ஏற்கனவே சொன்னதுப்போல, array காலியாக உள்ளதா என்பதை பின்வருமாறு கண்டுப்பிடிக்கலாம்.

```
days_of_week.empty?
=> true
```

Array class-யிலுள்ள size method-டை பயன்படுத்தி array-யின் அளவை கண்டுப்பிடிக்கலாம்:

```
days_of_week = Array.new(7)
days_of_week.size
=> 7
```



10.5Array கூறுகளை அணுகுதல்:

Array-யின் கூறுகளை அணுக (access) கூறுகளின் index மற்றும் [] செயற்கூறுகளைப் பயன்படுத்த வேண்டும். Array-யின் முதல் மற்றும் இரண்டாவது கூறுகளை access செய்ய,

```
days_of_week[0]
=> "Mon"

days_of_week[1]
=> "Tues"
```

இதேப்போல் array class-யிலுள்ள at செயற்கூற்றை பயன்படுத்தி அணுகலாம்,

```
days_of_week.at(0)
=> "Mon"
```

Array index -1-னை பயன்படுத்தி array-யின் கடைசி உருப்படியை அணுகலாம். உதாரணத்திற்கு,

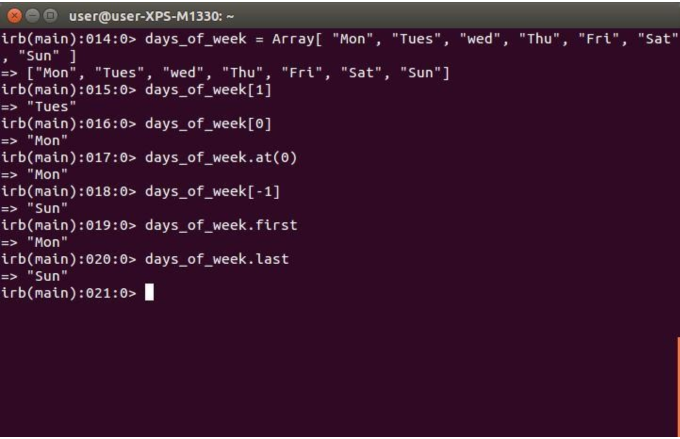
```
days_of_week[-1]
```

```
=> "Sun"
```

Array class-யிலுள்ள first மற்றும் last செயற்கூற்றை பயன்படுத்தி array-யின் முதல் மற்றும் கடைசி உருப்படிகளை அணுக முடியும்.

```
days_of_week.first
=> "Mon"

days_of_week.last
=> "Sun"
```



10.6 கூறுகளின் index-ஐக் கண்டறிதல்:

Index செயற்கூற்றை பயன்படுத்தி *array*-யின் குறிப்பிட்ட கூற்றின் *index*-ஐக் கண்டறியலாம். *Index* செயற்கூறானது பொருந்தும் முதல் கூற்றின் *index*-ஐத் திருப்பி அனுப்பும். உதாரணத்திற்கு நமது *days_of_week array*-யிலுள்ள "*wed*" கூறின் *index*-ஐக் கண்டுபிடிக்கலாம்.

```
days_of_week.index("Wed")
=> 2
```

Rindex method-டை பயன்படுத்தி *array*-யிலுள்ள

பொருந்தும் கடைசி கூற்றினை கண்டுபிடிக்கலாம்.

```
a = [1, 2, 3, 4, 5, 4, 3, 2, 1]
a.rindex("2")
=> 7
```

துணைக்குழுக்கள்

Array's கூறுகளில் துணைக்குழுவை எடுக்க ஆரம்ப எண் மற்றும் எத்தனை கூறுகள் எடுக்க வேண்டுமோ அதையும் கொடுக்க வேண்டும். உதாரணத்திற்கு ஆரம்ப கூறு 1 முதல் 3 கூறுகளையை எடுக்க,

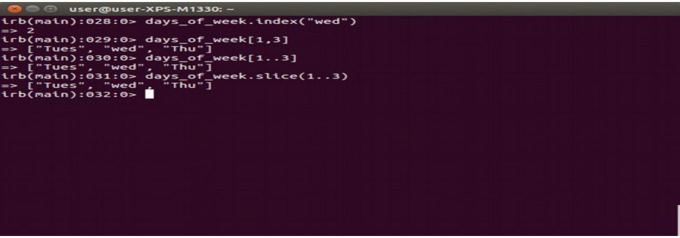
```
days_of_week[1, 3]
=> ["Tues", "Wed", "Thu"]
```

அதே போல், range-இலும் கொடுக்க முடியும்.

```
days_of_week[1..3]
=> ["Tues", "Wed", "Thu"]
```

மாற்றாக, array class-யிலுள்ள slice method-டையும் பயன்படுத்தலாம்,

```
days_of_week.slice(1..3)
=> ["Tues", "Wed", "Thu"]
```



11Advanced ரூபி arrays:

முந்தைய அத்தியாயத்தில் ரூபி array-யின் அறிமுகம் பார்த்தோம். இந்த அத்தியாயத்தில் விரிவாக பார்க்கலாம்.

11.1 ரூபி arrays இணைத்தல்:

ரூபியில் arrays-களை இணைக்க பல்வேறு அணுகுமுறைகளை பயன்படுத்தலாம். அதில் முதலவதாக கூட்டலை (+) பயன்படுத்தி இணைக்கலாம்,

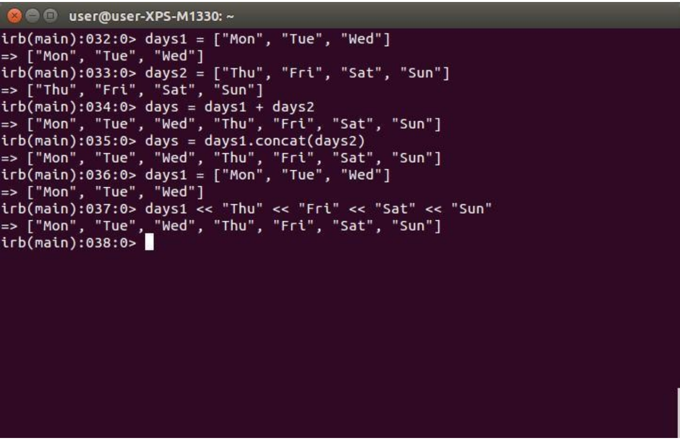
```
days1 = ["Mon", "Tue", "Wed"]
days2 = ["Thu", "Fri", "Sat", "Sun"]
days = days1 + days2
=> ["Mon", "Tue", "Wed", "Thu", "Fri",
"Sat", "Sun"]
```

மாற்றாக concat செயற்கூற்றையும் பயன்படுத்தலாம்.

```
days1 = ["Mon", "Tue", "Wed"]
days2 = ["Thu", "Fri", "Sat", "Sun"]
days = days1.concat(days2)
```

<< செயற்கூற்றை பயன்படுத்தி இருக்கும் array-யில் கூறுகளை இறுதியில் சேர்க்கலாம். உதாரணத்திற்கு,

```
days1 = ["Mon", "Tue", "Wed"]
days1 << "Thu" << "Fri" << "Sat" << "Sun"
=> ["Mon", "Tue", "Wed", "Thu", "Fri",
"Sat", "Sun"]
```



11.2Intersection, union மற்றும் difference:

ரூபியில Array-ஐ பலவகையில் கையாளலாம். Union, intersection மற்றும் difference பயன்படுத்தி இரண்டு array-களிலிருந்து ஒரு புது array-யை உருவாக்கலாம்.

Operator	*Description*
-	*Difference* - ஒரு புது *array*-யை திருப்பி அனுப்பும். முதல் *array*-யிலிருந்து, இரண்டாவது *array*-யிலுள்ள கூறுகளை நீக்கும்.
&	*Intersection* - ஒரு புது *array*-யை உருவாக்கி அதில் இரண்டு *array* -களின் பொதுவான கூறுகளை வைக்கும். நகல்களை நீக்கும்.
|	*Union* - இரண்டு *array*-களை இணைக்கும். நகல்களை நீக்கும்.

ஒரு சில எடுத்துக்காட்டுகள் *set operation*-யை விளக்க உதவும். இதற்கு பின்வருமாறு இரண்டு *array*-யை எடுத்து கொள்ளலாம்.

```
operating_systems = ["Fedora", "SuSE",
"RHEL", "Windows", "MacOS"]

linux_systems = ["RHEL", "SuSE",
"PCLinuxOS", "Ubuntu", "Fedora"]
```

இப்பொழுது இந்த இரண்டு *array*-யை *union* செய்து ஒரு புது *array*-யை உருவாக்கலாம்.

```
operating_systems | linux_systems
=> ["Fedora", "SuSE", "RHEL", "Windows",
"MacOS", "PCLinuxOS", "Ubuntu"]
```

மேலே உள்ள விடையில் ஒரு *array*-யை இன்னொரு *array*-வுடன் இணைத்து அதிலுள்ள நகல் *array* கூறுகளை நீக்குகிறது.

அடுத்ததாக *intersection* செய்யலாம்.

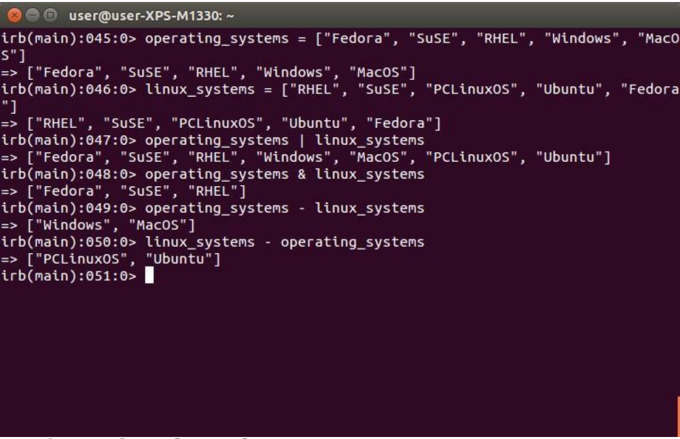
```
operating_systems & linux_systems
=> ["Fedora", "SuSE", "RHEL"]
```

இது இரண்டு *arrays*-யிலும் பொதுவாக உள்ள கூறுகளை விடையாக கொடுக்கும்.

இறுதியாக “*difference*” *operation*-னை பார்ப்போம்.

```
operating_systems - linux_systems
=> ["Windows", "MacOS"]
```

இரண்டு *array*-களின் *difference*-யை ஒரு புது *array*-யில் வைக்கும். நமது எடுத்துக்காட்டில்,*operating_systems*-ல் உள்ள கூறுகளிலிருந்து *linux_systems*-ல் உள்ள கூறுகளை



நீக்கிவிடும். இதை மேலும் விளக்க *operands* மாற்றம் செய்துப் பார்க்கலாம்,

```
linux_systems - operating_systems
=> ["PCLinuxOS", "Ubuntu"]
```

11.3 தனித்த array கூறுகளை கண்டறிதல்:

Array class-யிலுள்ள uniq method-டை கொண்டு நகல் array கூறுகளையை array-யிலிருந்து நீக்கலாம். உதாரணத்திற்கு,

```
linux_systems = ["RHEL", "SuSE",
"PCLinuxOS", "Ubuntu", "Fedora", "RHEL",
"SuSE"]

linux_systems.uniq
=> ["RHEL", "SuSE", "PCLinuxOS", "Ubuntu",
"Fedora"]
```

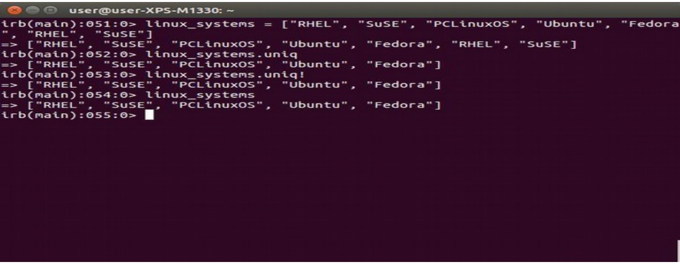
மேலே உள்ள உதாரணத்தில், uniq method-ஆல் அசல் array-யில் மாற்றம் எதுமில்லை. Uniq! Method-டை பயன்படுத்தி array-யிலிருந்து நகலை நீக்க முடியும், அதை பின்வருமாறு காணலாம். இது ரூபியில் பின்பற்றப்படும் ஒரு பொதுவான வழக்கமாகும் (common convention). எந்த ஒரு method-ம், அது எந்த object-ன் மீது அழைக்கப்டுகிறதோ, அதை மாற்றம் செய்யுமெனில், அதன் பெயர் ஆச்சரியகுறி கொண்டு முடிவடையவேண்டும்.

```
linux_systems
=> ["RHEL", "SuSE", "PCLinuxOS", "Ubuntu",
```

```
"Fedora", "RHEL", "SuSE"]

linux_systems.uniq!
=> ["RHEL", "SuSE", "PCLinuxOS", "Ubuntu",
"Fedora"]

linux_systems
=> ["RHEL", "SuSE", "PCLinuxOS", "Ubuntu",
"Fedora"]
```



11.4Array கூறுகளில் தள்ளுதல் மற்றும் மேலெடுத்தல்:

ரூபியின் array-யில் கூறுகளை தள்ளுதல் மற்றும் **மே***லெடுத்தல் செய்ய Last In First Out(LIFO) stack-யை பயன்படுத்துகிறது. இதை push மற்றும் pop methods-டை*

கொண்டு செய்யலாம், உதாரணத்திற்கு ஒரு *array* உருவாக்கி கூறுகளை உள்ளே தள்ளலாம்.

```
colors = ["red", "green", "blue"]
=> ["red", "green", "blue"]

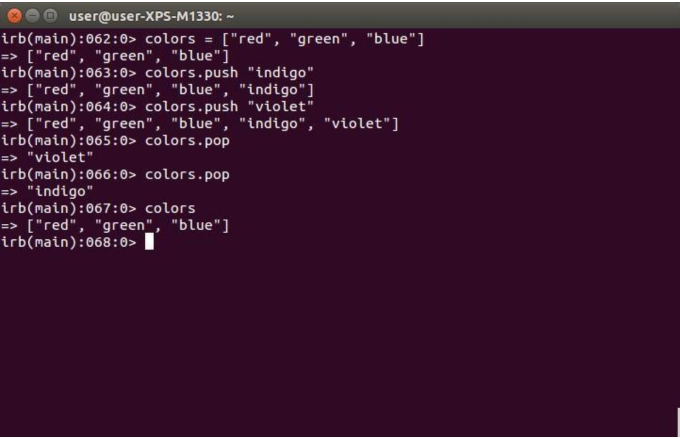
colors.push "indigo"
=> ["red", "green", "blue", "indigo"]

colors.push "violet"
=> ["red", "green", "blue", "indigo",
"violet"]
```

Pop method-டை பயன்படுத்தி *array*-யிலிருந்து கூறினை வெளியே எடுக்கலாம்.

```
colors.pop
=> "violet"

colors.pop
=> "indigo"
```



11.5 ரூபி array ஒப்பீடுகள்:

ரூபி *arrays*-யை *==,<=>* மற்றும் *eql? Method*-டை பயன்படுத்தி ஒப்பிடலாம்.

இரண்டு *array*-யிலும் ஒரே எண்ணிக்கையில் கூறுகளையும் மற்றும் ஒத்த இடத்திலுள்ள கூறுகளிலும் *(corresponding elements)* ஒரே *content*-ம் இருந்தால் *==* *method true*-வை திருப்பியனுப்பும்.

Eql? Method, == method போன்றதுதான். ஆனால்,

இரண்டு *arrays*-யிலும் உள்ள *corresponding* கூறுகளின் வகையும் *(value type)* ஒன்றாக இருக்க வேண்டும்.

இறுதியாக <=> *method*, இதை "*spaceship*" *method* என்றும் அழைக்கலாம். இது இரண்டு *array*-யை ஒப்பீடு செய்து *equal* என்றால் 0-வை திருப்பி அனுப்பும். ஒரு *Array* கூறுகள் மற்றொரு *array* கூறுகளைவிட குறைவாக இருந்தால் -1-னையும் கூடுதலாக இருந்தால் 1-னையும் திருப்பி அனுப்பும்.

```
2.2.0 :031 > a = [1.0, 2.0, 3.0, 4.0, 5.0]
 => [1.0, 2.0, 3.0, 4.0, 5.0]
2.2.0 :032 > b = [1, 2, 3, 4, 5]
 => [1, 2, 3, 4, 5]
2.2.0 :033 > a == b
 => true
2.2.0 :034 > a.eql? b
 => false
2.2.0 :035 >
```

11.6Arrays மாற்றியமைத்தல்:

Array-யில் இடையில் ஒரு புது கூறை செருக insert method-டை பயன்படுத்த வேண்டும். செருக வேண்டிய கூறின் index மதிப்பு மற்றும் புது மதிப்பையும் இந்த method-க்கு argument ஆக கொடுக்க வேண்டும், உதாரணத்திற்கு ஒரு புதிய வண்ணத்தை red மற்றும் green கூற்றிற்கு நடுவில் செருக பின்வருமாறு செய்யலாம்.

```
colors = ["red", "green", "blue"]
=> ["red", "green", "blue"]

colors.insert( 1, "orange" )
=> ["red", "orange", "green", "blue"]
```

array கூறையும் array கூறின் index-யை பயன்படுத்தி ஒரு புது மதிப்பை கொடுக்க முடியும்.

```
colors = ["red", "green", "blue"]
```

```
=> ["red", "green", "blue"]

colors[1] = "yellow"
=> "yellow"

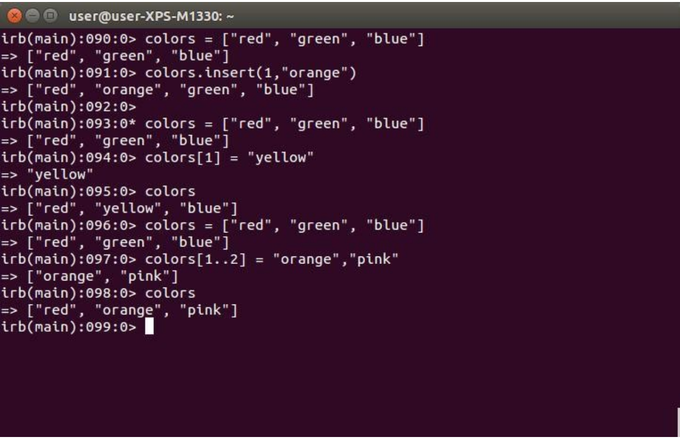
colors
=> ["red", "yellow", "blue"]
```

Range-ஐ பயன்படுத்தி பல கூறினையை மாற்ற முடியும்.

```
colors = ["red", "green", "blue"]
=> ["red", "green", "blue"]

colors[1..2] = "orange", "pink"
=> ["orange", "pink"]

colors
=> ["red", "orange", "pink"]
```



11.7Array-யிலிருந்து கூறுகளை நீக்குதல்:

Array-யிலிருந்து கூறுகளை, ஒன்று array கூறின் content-யையோ அல்லது index இருப்பினை கொண்டோ நீக்கலாம்.

Index-யை பயன்படுத்தி நீக்க delete_at method-டை பயன்படுத்தலாம்:

```
colors = ["red", "green", "blue"]
=> ["red", "green", "blue"]
```

```
colors.delete_at(1)
=> "green"

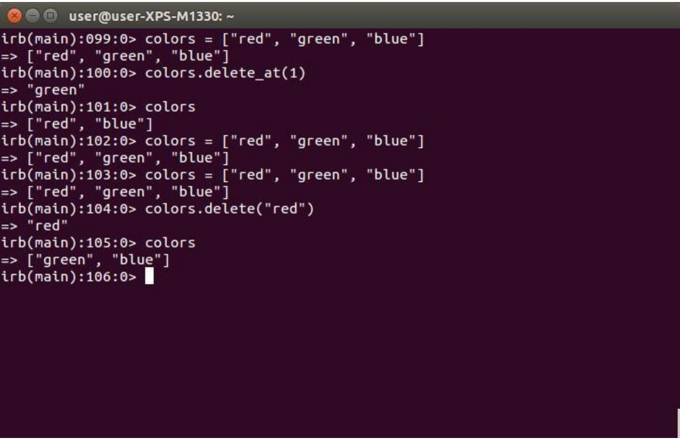
colors
=> ["red", "blue"]
```

Array கூறின் content-டை கொண்டு நீக்க delete method-டை பயன்படுத்தலாம்.

```
colors = ["red", "green", "blue"]
=> ["red", "green", "blue"]

colors.delete("red")
=> nil

colors
=> ["green", "blue"]
```



11.8Arrays வரிசைப்படுத்துதல்:

ரூபியில் arrays வரிசைப்படுத்த sort மற்றும் reverse method-ஐ பயன்படுத்த வேண்டும்.

```
numbers = [1, 4, 6, 7, 3, 2, 5]
=> [1, 4, 6, 7, 3, 2, 5]

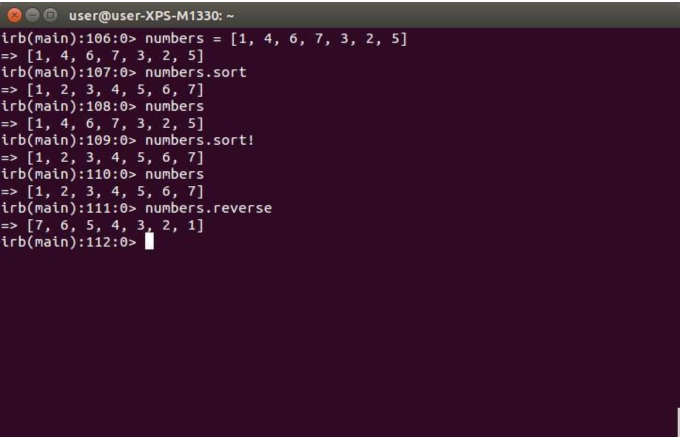
numbers.sort
=> [1, 2, 3, 4, 5, 6, 7]
```

அசல் *array*-யை வரிசைப்படுத்த *sort! Method*-டை பயன்படுத்த வேண்டும். *Array* கூறினை வரிசையை மாற்ற *reverse method*-டை பயன்படுத்தி செய்யலாம்.

```
numbers = [1, 4, 6, 7, 3, 2, 5]
=> [1, 4, 6, 7, 3, 2, 5]

numbers.sort!
=> [1, 2, 3, 4, 5, 6, 7]

numbers.reverse
=> [7, 6, 5, 4, 3, 2, 1]
```


user@user-XPS-M1330: ~
irb(main):106:0> numbers = [1, 4, 6, 7, 3, 2, 5]
=> [1, 4, 6, 7, 3, 2, 5]
irb(main):107:0> numbers.sort
=> [1, 2, 3, 4, 5, 6, 7]
irb(main):108:0> numbers
=> [1, 4, 6, 7, 3, 2, 5]
irb(main):109:0> numbers.sort!
=> [1, 2, 3, 4, 5, 6, 7]
irb(main):110:0> numbers
=> [1, 2, 3, 4, 5, 6, 7]
irb(main):111:0> numbers.reverse
=> [7, 6, 5, 4, 3, 2, 1]
irb(main):112:0>

12 ரூபி செயற்குறிகள்:

இந்த அத்தியாயத்தில் ரூபியின் expressions உருவாக்க பயன்படும் operators-ன் அடிப்படைகளை காணலாம். ரூபியில் பல்வேறு செயற்குறிகள் (operators) உள்ளன.

- *Assignment Operators*
- *Math Operators*
- *Comparison Operators*
- *Bitwise Operators*

12.1 ரூபி செயல்பாடுகள்:

எந்த மதிப்பை கொண்டு கணக்கீடு செய்யபடுகிறதோ அது செயலேற்பி (operand) ஆகும். கணக்கீடு செய்ய பயன்படுவதை செயற்குறிகள் (operators) எனலாம். செயற்குறிகளின் இரு பக்கமும் செயலேற்பிகள் இருக்கும். செயல்பாட்டின் விடையை assignment operator(=)-ரை பயன்படுத்தி ஒரு மாறிக்கு வழங்க வேண்டும். Irb-யில் பெரும்பாலான அடிப்படை செயல்பாடுகளை (operation) செயல்படுத்த முடியும்.

```
1 + 1
=> 2
```

இப்பொழுது, “result” என்னும் மாறியில் விடையை வைக்கலாம்.

```
result = 1 + 1
=> 2
```

12.2 ரூபியில் எண்கணித செயல்பாடுகள்:

ரூபியில் கணித செயல்பாடுகளுக்கென(arithmetic operations) பல அடிப்படையான செயற்குறிகள் உள்ளன, அவை பின்வருமாறு.

Operator	*Description*
+	*செயற்குறியின் இருபக்கம் உள்ள மதிப்பினை கூட்டும்*
-	*இடது கை செயலேற்பியிலிருந்து வலது கை செயலேற்பியை*

	கழிக்கும்
*	செயற்குறியின் இருபக்கம் உள்ள மதிப்பினை பெருக்கும்
/	வலது கை செயலேற்பியால் இடது கை செயலேற்பியை வகுக்கும்.
%	*Modulus* - வலது கை செயலேற்பியால் இடது கை செயலேற்பியை வகுத்து மீதியை திருப்பி அனுப்பும்
**	*Exponent - exponential (power)* கணக்கீடு செய்யும்

இரண்டு *integer*-களை வகுக்கும் பொழுது, விடையும், ஒரு *integer* -ஆகவே இருக்கும். விடையை *truncate* செய்யாமல், தசம இலக்கங்களுடன், முழு எண்ணாக வேண்டுமெனில், செயல்பாட்டின் குறைந்தது ஒரு செயலேற்பியாவது *float*-ஆக கொடுக்க வேண்டும்.

```
10 / 7
=> 1
```

மேலே உள்ள எடுத்துக்காட்டில் விடை ஆனது அருகிலுள்ள முழு எண்ணாக ஆக்கப்பட்டுள்ளது. ஒரு செயலேற்பியை float-ஆக கொடுத்தால், மிக சரியான பதிலை பெறலாம்.

```
 10.0 / 7
=> 1.42857142857143
```

12.3 ரூபி assignment operators:

assignment operator-ஆனது, ஒரு மாறிக்கு மதிப்பினை வழங்க பயன்படுகிறது. எடுத்துக்காட்டாக, இரு எண்களை கூட்டி, அதன் மதிப்பை ஒரு மாறிக்கு வழங்கவேண்டுமெனில், பின்வருமாறு செயற்குறிகளை பயன்படுத்தலாம்.

```
x = 2
y = 3
z = x + y # 5
```

ஒரு கணித செயல்பாட்டின் விடையை, அதன்

செயலேற்பிகளுள் ஒன்றிற்கு வழங்குவது, சாதாரணமாக நிரலாலர்களிடம் காணப்படும் வழக்கமாகும். இதனை பின்வருமாறு எழுதலாம்.

```
x = 2
y = 3
x = x + y # 5
```

இதை சுருக்கமாக செய்வதற்கு ரூபியில், சில எளிய செயற்குறிகள் உண்டு. உதாரணமாக,

```
x = 2
y = 3
x += y # 5
```

இதில் பல்வேறு *assignment operators* உள்ளது. மாறியில் மதிப்பை வைப்பதற்கு முன்னரே, செயலேற்பியின் கணக்கீட முடிந்து விடுகிறது. இது எண்கணிதம் மற்றும் *assignment operator* இணைந்ததாகும். இந்த வகையிலுள்ள பொதுவான செயற்குறிகள் பின்வருமாறு.

Combined Operator	*Equivalent*
x += *y*	*x* = *x* + *y*
x -= *y*	*x* = *x* - *y*
x /= *y*	*x* = *x* / *y*
x *= *y*	*x* = *x* * *y*
x %= *y*	*x* = *x* % *y*
x **= *y*	*x* = *x* ** *y*

இந்த இணைந்த செயற்குறிகளைக்கொண்டு, இரு மாறிகளின் எண்கணித செயல்பாட்டின் விடையை எளிய வழியில் கொடுக்கலாம். இரு *variable*-லின் கணக்கீடு

விடை ஆனது முதல் மாறியில் வைக்கப்படுகிறது. உதாரணத்திற்கு,

```
x = 10
x += 5 # variable x-ல் மதிப்பு 15 வைக்கப்படுகிறது
(x = x + 5 க்கு இணையானது)

y = 20
y -= 10  # variable y-ல் மதிப்பு 10
வைக்கப்படுகிறது (y = y – 10 க்கு இணையானது)

x = 10
y = 5
x /= y # variable x-ல் மதிப்பு 2 வைக்கப்படுகிறது
(x = x / y க்கு இணையானது)
```

12.4 இணை assignment:

ரூபியில், variable-க்கு இணை (parallel) assignment செய்ய முடியும். பல மாறிகளுக்கு ஒரு வரியிலேயே தொடக்க மதிப்பை கொடுக்க முடியும்.

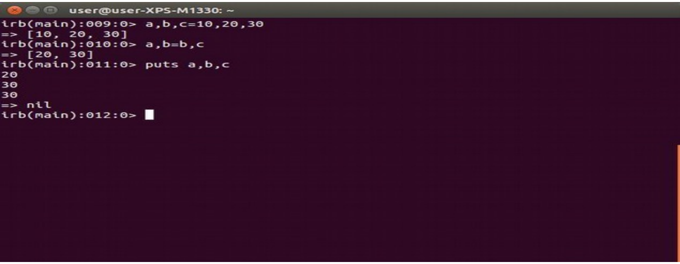
```
a = 10
b = 20
c = 30
```

இணை assignment-டை பயன்படுத்தி மாறிகளுக்கு தொடக்க மதிப்பை கொடுக்கலாம்.

```
a, b, c = 10, 20, 30
```

இணை assignment-டை பயன்படுத்தி இரு மாறிகளிலிலுள்ள மதிப்பை இடமாற்றம் செய்யவும் முடியும்.

```
a, b = b, a
```



12.5 ரூபி ஒப்பீட்டு செயற்குறிகள்:

இரு மதிப்புகளின் சமநிலையை சரிப்பார்க்க ஒப்பீட்டு செயற்குறிகள் (comparison operators) பயன்படுத்தலாம். இதற்கு ரூபியில் பல செயற்குறிகள் உள்ளன:

Comparison Operator	*Description*
==	*சமத்தன்மையை சரிபார்க்கும். True அல்லது false-யை திருப்பி அனுப்பும்.*
.eql?	*இது == போன்றதுதான்.*
!=	*சமனின்மையை சரிப்பார்க்கும். இரு மதிப்புகள் சமனில்லையென்றால் true-வையும், சமமானதென்றால் false-யையும் திருப்பி அனுப்பும்.*
<	*இரண்டாவது செயலேற்பியை விட முதல் செயலேற்பி சிறியதாக இருந்தால்true-வையும் இல்லையென்றால் false-யை*

	திருப்பி அனுப்பும்.
>	இரண்டாவது செயலேற்பியை விட முதல் செயலேற்பி பெரியதாக இருந்தால் *true*-வையும் இல்லையென்றால் *false*-யை திருப்பி அனுப்பும்.
>=	இரண்டாவது செயலேற்பியை விட முதலாவது செயலேற்பி பெரியதாகவோ அல்லது சமமாகவோ இருந்தால் *true*-வையும் அல்லது *false*-யையும் திருப்பி அனுப்பும்.
<=	இரண்டாவது செயலேற்பியை விட முதல் செயலேற்பி சிறியதாகவோ அல்லது சமமாகவோ இருந்தால் *true*-வையும் அல்லது *false*-யையும் திருப்பி அனுப்பும்.
<=>	இரண்டு செயலேற்பிகளும் சமமாக இருந்தால் *0*-வை திருப்பி அனுப்பும், முதல்

	செயலேற்பியானது இரண்டாவது செயலேற்பியை விட பெரியதாக இருந்தால் 1-னையும், முதல் செயலேற்பி சிறியதாக இருந்தால் -1-னையும் திருப்பி அனுப்பும்.

சில எடுத்துக்காட்டுகளை பார்ப்போம்.

```
1 == 1 # true

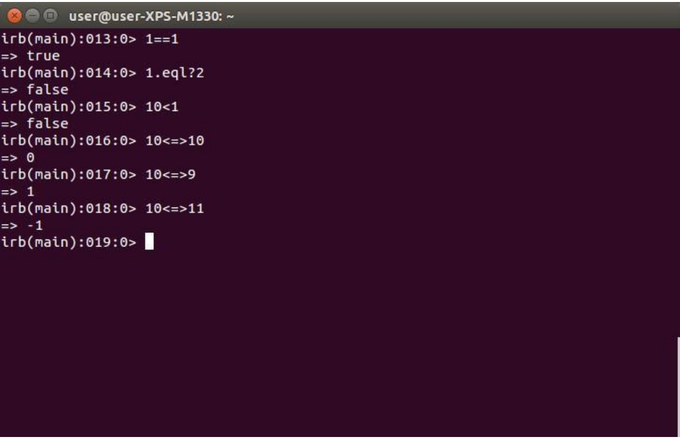
1.eql? 2 # false

10 < 1 # false

10 <=> 10 # 0

10 <=> 9 # 1

10 <=> 11 # -1
```



12.6 ரூபி Bitwise operators:

எண்களை *bit* அளவில் கணக்கீடுகள் செய்ய *Bitwise operators* பயன்படுத்தலாம். கணினி எல்லாவற்றையும் *binary* மதிப்பாகவே பார்க்கும் *(1* மற்றும் *0).*

உதாரணத்திற்கு, கணினி *520* எண்ணை *01010 binary*

மதிப்பாக பார்க்கும்.

Combined Operator	*Equivalent*
~	*Bitwise NOT (Complement)*
|	*Bitwise OR*
&	*Bitwise AND*
^	*Bitwise Exclusive OR*
<<	*Bitwise Shift Left*
>>	*Bitwise Shift Right*

ருபியில் கணித *operator*-களைப்போல, *bitwise operator*-களையும், *assignment operator*-உடன் இணைத்து பயன்படுத்தலாம் *(* ~=, >>=, <<= ^=, &=*).*

13 ரூபி செயற்குறிகளின் முன்னுரிமை:

முந்தைய அத்தியாயத்தில் ரூபி செயற்குறிகள் மற்றும் expressions-யை பார்த்தோம். அதற்கு இணையாக செயற்குறிகளின் முன்னுரிமையை (precedence) புரிந்து கொள்ளுதல் அவசியமாகும். ஒன்றுக்கு மேற்பட்ட செயற்குறிகள் உள்ள expression-னை ரூபி interpreter எந்த வரிசையில் மதிப்பீடு செய்யும் என்பதை முன்னுரிமை நிர்ணயிக்கிறது.

13.1 எடுத்துக்காட்டு:

நாம் expressions இடது பக்கத்திலிருந்து வலது பக்கமாக மதிப்பீடு செய்வோம். உதாரணத்திற்கு, பின்வரும் expression-னை இடது பக்கம் முதல் வலது பக்கமாக மதிப்பீடு செய்தால் விடை 300 என வரும்:

*10 + 20 * 10 = 300*

இடது பக்கத்திலுள்ள 10-யும் 20-தையும் கூட்டி வரும் 30-தை 10-வுடன் பெருக்கினால் 300 வரும். இதையே ரூபியில் மதிப்பீடல் செய்தால், முற்றிலும் வேறு விடையை தரும்.

```
10 + 20 * 10
```

```
=> 210
```

ரூபியில் ஒரு expression-னிலுள்ள செயற்குறிகளை மதீப்பிடு செய்ய சில விதிமுறைகள் உண்டு. ரூபியில் கூட்டலுக்கான செயற்குறியை (+) விட பெருக்கல் செயற்குறி (*) அதிக முன்னுரிமையை கொண்டது.

13.2Overriding operator முன்னுரிமை:

ஒரு expression-னில் குறைந்த முன்னுரிமை உள்ள பகுதியை அடைப்புக்குறிக்குள் கொடுத்து அதிக முன்னுரிமை உள்ளதாக ஆக்கலாம் உதாரணத்திற்கு,

```
(10 + 20) * 10
=> 300
```

மேலே உள்ள எடுத்துக்காட்டில், அதிக முன்னுரிமை உள்ள பெருக்கல் குறிக்கு (*) முன்னர் அடைப்பிலுள்ள மதிப்பை மதிப்பீடு செய்யும்.

13.3 செயற்குறி முன்னுரிமை அட்டவணை:

பின்வரும் அட்டவணையில், உயர்ந்த

முன்னுரிமையிலிருந்து குறைந்த முன்னுரிமை வரையிலான செயற்குறிகள் பட்டியலிடப்பட்டுள்ளன.

Method	*Operator*	*Description*
ஆம்	*[ ] [ ]=*	உருப்படியை அணுக *(element reference)*, உருப்படிக்கு மதிப்பளிக்க *(set element)*
ஆம்	**	அடுக்குக்குறி *(Exponentiation - raise to the power)*
ஆம்	! ~ + -	*Not, complement, unary plus and minus (method names for the last*

		two are +@ and -@)
ஆம்	* / %	பெருக்கல், வகுத்தல், மீதி
ஆம்	+ −	கூட்டல் மற்றும் கழித்தல்
ஆம்	>> <<	*Right and left bitwise shift*
ஆம்	&	*Bitwise 'AND'*
ஆம்	^ |	*Bitwise exclusive 'OR' and regular 'OR'*
ஆம்	<= < > >=	ஒப்பீட்டு செயற்குறிகள்

<table>
<tr><td>ஆம்</td><td><=> == === ! = =~ !~</td><td>Equality and pattern match operators (!= and !ஸ்ரீ may not be defined as methods)</td></tr>
<tr><td></td><td>&&</td><td>Logical ‘AND’</td></tr>
<tr><td></td><td>||</td><td>Logical ‘AND’</td></tr>
<tr><td></td><td>.. and ...</td><td>Range (inclusive and exclusive)</td></tr>
<tr><td></td><td>? :</td><td>Ternary if-then-else</td></tr>
<tr><td></td><td>= %= { /= -= += |= &= >>= <<= *= &&= || = **=</td><td>Assignment</td></tr>
</table>

	defined?	*Check if specified symbol defined*
	Not	*Logical negation*
	or and	*Logical composition*
	if unless while until	*Expression modifiers*
	begin/end	*Block expression*

13.4Overriding an operator:

மேலே உள்ள அட்டவணையிலுள்ள செயற்கூறு என்ற பத்தியில் “ஆம்” ஆக இருந்தால் அதை *override* செய்யலாம்.

உதரணத்திற்கு, << செயற்குறியை எடுத்துக்கொள்ளலாம்.

இந்த செயற்குறியின் இயல்பான பயன்பாட்டை கீழேகாணலாம்.

```
'Hello' << 'World'
=> "HelloWorld"
```

இதை override செய்ய முயற்சிக்கலாம். இரண்டாவது string-ஐ முதல் string-ன் பின்னால் இணைப்பதற்கு (suffix) பதிலாக, முன்னொட்டாக (prefix) இணைக்கவேண்டுமென்றால், << செயற்குறியை பின்வருமாறு override செய்யவேண்டும்.

```
class String
  def <<(value)
    self.replace(value + self)
  end
end

'Hello' << 'World'

=> "WorldHello"
```

```
2.2.0 :005 > 'Hello' << 'World'
 => "HelloWorld"
2.2.0 :006 > class String
2.2.0 :007?>     def <<(value)
2.2.0 :008?>         self.replace(value + self)
2.2.0 :009?>       end
2.2.0 :010?>   end
 => :<<
2.2.0 :011 > 'Hello' << 'World'
 => "WorldHello"
2.2.0 :012 >
```

14 ரூபி கணித செயற்கூறுகள்

ரூபியில் கணித கூறானது (math module) நிரலருக்கு பல செயற்கூறுகளைக் கொடுக்கிறது. இதை கொண்டு பல கணக்கீடுகள் செய்ய முடியும். கூடுதலாக இதில் இரண்டு பொதுவான □□□□□□கள் (mathematical constants) உள்ளன.

14.1 ரூபி கணித மாறிலிகள்:

கணித கூற்றில் உள்ள மாறிலிகளை, Constants என்ற செயற்கூற்றை பயன்படுத்தி, பட்டியலிடலாம்.

```
Math.constants
=> ["E", "PI"]
```

ரூபியின் தற்போதைய பதிப்பின்படி இரண்டு மாறிலிகளே வரையறுக்கப்பட்டுள்ளது. இதை :: குறியீட்டை பயன்படுத்தி அணுகலாம்.

```
Math::PI
=> 3.14159265358979

Math::E
=> 2.71828182845905
```

14.2 ரூபி கணித செயற்கூறுகள்

ரூபியில் பலவகையாக கணித செயற்கூறுகள் உள்ளன. அதை பின்வரும் அட்டவணையில் காணலாம்.

Method name	*Description*
Math.acos, Math.acos!	*Arc cosine*
Math.acosh, Math.acosh!	*Hyperbolic arc cosine*
Math.asin, Math.asin!	*Arc sine*
Math.asinh, Math.asinh	*Hyperbolic arc sine*
Math.atan,Math.atan!, Math.atan2, Math.atan2!	*Arc tangent. atan takes an x argument. atan2 takes x and y arguments*

Math.atanh, Math.atanh!	*Hyperbolic arc tangent*
Math.cos, Math.cos!	*Cosine*
Math.cosh, Math.cosh	*Hyperbolic cosine*
Math.sin, Math.sin!	*Sine*
Math.erf	*Error function*
Match.erfc	*Complementary error function*
Math.exp, Math.exp!	*Base x of Euler*
Math.frexp	*Normalized fraction and exponent*

Math.hypot	*Hypotenuse*
Math.ldexp	*Floating-point value corresponding to mantissa and exponent*
Math.sinh, Math.sinh!	*Hyperbolic sine*
Math.sqrt, Math.sqrt!	*Square root*
Math.tan, Math.tan!	*Tangent*
Math.tanh, Math.tanh!	*Hyperbolic tangent*

14.3 சில எடுத்துக்காட்டுகள்:

இப்போது கணித செயற்கூறுகளை எப்படி பயன்படுத்துவது என பார்க்கலாம்.

வர்க்கமூலத்தை கண்டுப்பிடிக்க,

```
Math.sqrt(9)
=> 3.0
```

அல்லது Euler calculation:

```
Math.exp(2)
=> 7.38905609893065
```

14.4 ரூபி தர்க்க செயற்குறிகள்:

தர்க்க செயற்குறிகளை (logical operators), Boolean operators என்றும் சொல்லலாம். ஏனென்றால் இது expression மதிப்பிட்டு true அல்லது false-யை திருப்பி அனுப்பும். முதல் படியாக, இவ்வகை செயற்குறிகள் ரூபியில் எவ்வாறு வேலை செய்கிறது என்பதை அறிய, நிரல் எழுதுவதற்கு முன்பாக, ஒரு சொற்றொடரை எழுதிப்பார்க்கலாம். var1, var2 என்று இரு மாறிகளை எடுத்துக்கொண்டு இதை முயற்சித்துப்பார்க்கலாம்:

If var1 is less than 25 AND var2 is greater than 45 return true.

இதில்,"AND" என்பதே தர்க்க செயற்குறி ஆகும். இந்த expression-னை ரூபியில், ஒப்பீட்டு (comparison operator)

மற்றும் and அல்லது && என்ற தர்க்க செயற்குறிகளைப் பயன்படுத்தி எழுதலாம்.

```
var1 = 20
var2 = 60
var1 < 25 and var2 > 45
=> true
```

அதேபோல்,

If var1 is less than 25 OR var2 is greater than 45 return true.

இதில் "OR" பதிலாக ரூபியில் or அல்லது "||" பயன்படுத்த வேண்டும்.

```
var1 < 25 or var2 > 45
=> true
```

மற்றொரு தர்க்க செயற்குறி, not operator ஆகும். இது expression-னின் விடைக்கு எதிர்மறையான மதிப்பைத்தரும். NOT operator —க்கு ரூபியில் !குறியீட்டை பயன்படுத்த வேண்டும்.

```
10 == 10
=> true

!(10 == 10)         // ''false'' திருப்பி
```
அனுப்பும் ஏனென்றால் not operator கொண்டு விடை எதிர்மறையானதாக மாற்றப்படும்.

```
=> false
```

user@user-XPS-M1330: ~

```
irb(main):007:0> var1 = 20
=> 20
irb(main):008:0> var2 = 60
=> 60
irb(main):009:0> var1 < 25 and var2 > 45
=> true
irb(main):010:0> var1 < 25 or var2 > 45
=> true
irb(main):011:0> 10==10
=> true
irb(main):012:0> !(10==10)
=> false
irb(main):013:0>
```

15 ரூபியில் பொருள் நோக்கு நிரலாக்கம்:

ரூபி பொருள் நோக்கு பயன்பாடுகளை *(object oriented applications)* உருவாக்க ஏதுவான சூழலைத்தருகிறது. பொருள் நோக்கு நிரலாக்கம் பற்றிய களம் மிகவும் பெரியது. அதை பற்றிய முழுமையாக விளக்கத்தை அளிப்பது இந்த பதிவின் நோக்கமல்ல. ஆகையால் பொருள் நோக்கு நிரலாக்கத்தின் அடிப்படை கருத்துகளையும், ரூபி நிரலாக்கத்திற்கு தேவையான கருத்துகளையும் மட்டும் பார்க்கலாம்.

15.1 பொருள் என்றால் என்ன:

பொருள் *(Object)* என்பது எளிமையான, சிறு செயல்பாட்டினை, தன்னுள் கொண்டதாகும். இது பலமுறை பயன்படுத்தக்கூடியதாகவும், ஒரு மென்பொருளை நிர்மாணிக்க தேவையான அடிப்படை கூறாகவும் பயன்படுகிறது. பொருளில் தரவு மாறிகளும் *(data variable)*, செயற்கூறுகளும் *(function/method)* இருக்கும். இவை ஒரு பொருளின் உறுப்பினர்கள் *(members)* என அழைக்கப்படுகின்றன.

ரூபியில், எண்கள் முதல் *string, array* வரை எல்லாமே பொருட்கள் தான்.

15.2 **வர்க்கம் என்றால் என்ன?:**

கட்டிடம் கட்டியபிறகு எப்படி இருக்கும் என்பதை, ஒரு வரைபடம் விளக்குவதுபோல, *வர்க்கமானது (class), ஒரு பொருள் (object) எப்படி உருவாக்கப்படவேண்டும் என்பதை விளக்குகிறது. உதாரணத்திற்கு, செயற்கூறுகள் என்ன செய்யும், என்னென்ன மாறிகள் உறுப்பினர்களாக இருக்கும் என்பதை விளக்குவதாகும்.*

Inheritance என்பது ஏற்கனவே உள்ள வர்க்கத்தை அடிப்படையாகக்கொண்டு, புது வர்க்கத்தை உருவாக்குவதாகும். இதில் புது வர்க்கமானது (subclass) parent class (super class) லிருந்து எல்லா அம்சங்களையும் மரபுரிமையாகப்பெற்றிருக்கும். அதோடு sub class-யில், super class-யில்லாத செயல்பாடுகளையும் சேர்க்கலாம். ரூபி single inheritance-யை ஆதரிக்கிறது. அப்படியென்றால் subclass ஆனது ஒரேயொரு superclass-யிலிருந்து மட்டும் inherit செய்யப்படும்.
Java போன்ற மற்ற மொழிகள் multiple inheritance-யை ஆதரிக்கும். அதாவது subclass ஆனது ஒன்றுக்கு மேற்பட்ட superclasses-யை inherit செய்யும்.

15.3 **ரூபி வர்க்கத்தின் வரையறை:**

இந்த பயிற்சிகாக வங்கிப்பயன்பாட்டின், ஒரு பகுதியாக

ஒரு புதிய வர்க்கத்தை உருவாக்கலாம்.
வர்க்கங்களை *class* என்ற திறவுச்சொல் *(keyword)* கொண்டு வரையறுக்க வேண்டும். *end* என்ற திறவுச்சொல் கொண்டு முடிக்க வேண்டும்.
வர்க்கத்தை அதன் பெயரை கொண்டு அறிய வேண்டும்.
வர்க்கத்தின் பெயர் ஒரு மாறிலி *(constant)* ஆகும்.
வர்க்கத்தின் பெயரை *upper camel case* முறையில் எழுதவேண்டும்.

ஒரு வர்க்கத்தை பின்வருமாறு வரையறுக்கலாம்:

```
class BankAccount
   def initialize ()
   end

   def test_method
        puts "The class is working"
   end
end
```

மேலே உள்ள எடுத்துக்காட்டில், வர்க்கத்தின் பெயர் *BankAccount* ஆகும், அதில் *test_method* என்றொரு செயற்கூறு உள்ளது. இந்த செயற்கூற்றில் *string*-யை அச்சிடும் நிரல் உள்ளது.
மேலும் *initialize* செயற்கூறு, ரூபியின் ஒரு நிலையான *(standard)* "வர்க்க செயற்கூறு" *(class method)* ஆகும்.

வர்க்கத்திற்கு பொருட்களை உருவாக்கும் பொழுது *initialize* செயற்கூறு தான் முதலில் அழைக்கப்படும். இந்த செயற்கூற்றில் எந்த நிரலை வேண்டுமானாலும் எழுதலாம். *Java, C#* போன்ற மொழிகளிலுள்ள *constructor*-க்கு இணையாக இதைக்கருதலாம்

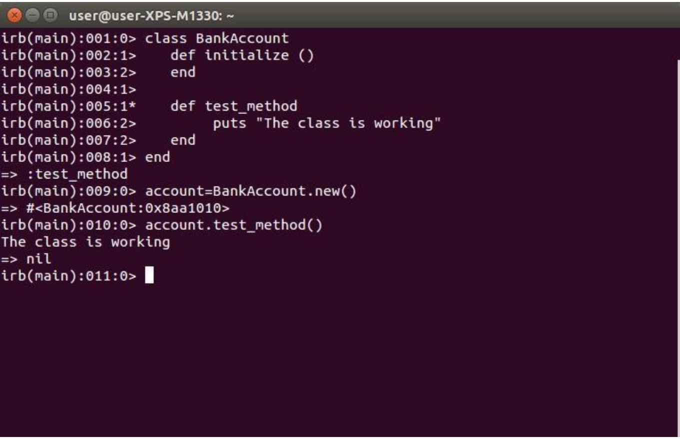
15.4 வர்க்கத்தின் பொருட்களை உருவாக்குதல்:

வர்க்கத்திலிருந்து ஒரு பொருளை உருவாக்க *new* என்ற செயற்கூற்றை பயன்படுத்த வேண்டும். உதாரணத்திற்கு,*BankAccount* என்கிற வர்க்கத்திற்கு *instance* உருவாக்க பின்வருமாறு எளிமையாக எழுதலாம்:

```
account = BankAccount.new()
```

இப்போழுது, *BankAccount*-ற்கு *account* எனும் பொருள் உருவாக்கப்பட்டுள்ளது. உருவாக்கிய பொருளைக்கொண்டு *test_method*-டை அழைக்கலாம்:

```
account.test_method
The class is working
```



15.5 உருபொருள் மாறிகளும், அணுக்க செயற்கூறுகளும்:

உருபொருள் மாறி (Instance variable) என்பது ஒரு மாறி. அது வர்க்கத்தில் வரையறுக்கப்பட்டிருக்கும், அது வர்க்கத்தின் ஒவ்வொரு உருபொருளுக்கும் கிடைக்ககூடியாக இருக்கும். இவ்வகை மாறிகளை வர்க்க செயற்கூறுகளின் (class methods) உள்ளேயோ அல்லது வெளியவோ வரையறுக்கலாம். பொதுவான இவை initialize செயற்கூற்றில் வரையறுக்கப்படும். மாறிகளை வர்க்கத்திற்கு வெளியில் பயன்படுத்துவதற்கு, அணுக்க

செயற்கூறுகளை (accessor methods/getter method) வரையறுக்க வேண்டும்.

உதாரணத்திற்கு, நமது BankAccount வர்க்கத்தில் உருபொருள் மாறிகளை சேர்க்க:

```
class BankAccount
   def initialize
             @accountNumber = "12345"
             @accountName = "John
Smith"
   end

   def accountNumber
        @accountNumber
   end

   def accountName
        @accountName
   end

   def test_method
        puts "The class is working"
        puts accountNumber
   end
end
```

இப்போழுது, @accountNumber மற்றும் @accountName இரண்டு உருபொருள் மாறிகள், அணுக்க

```
kalarani:intro kalarani$ irb
2.2.0 :001 > class BankAccount
2.2.0 :002?>   def initialize
2.2.0 :003?>     @account_number = 12345
2.2.0 :004?>     @account_name = "John Smith"
2.2.0 :005?>     end
2.2.0 :006?>   def account_number
2.2.0 :007?>     @account_number
2.2.0 :008?>     end
2.2.0 :009?>   def account_name
2.2.0 :010?>     @account_name
2.2.0 :011?>     end
2.2.0 :012?>   def test_method
2.2.0 :013?>     p "This class is working"
2.2.0 :014?>     p @account_number
2.2.0 :015?>     end
2.2.0 :016?>   end
 => :test_method
2.2.0 :017 > account = BankAccount.new
 => #<BankAccount:0x007fd0e4a3cf48 @account_number=12345, @account_name="John Smith">
2.2.0 :018 > account.account_number
 => 12345
2.2.0 :019 > account.account_name
 => "John Smith"
2.2.0 :020 >
```

செயற்கூறுகளுடன் இணைந்துள்ளது. இப்போழுது இந்த மாறிகளை நாம் வெளியிலிருந்து பயன்படுத்த முடியும்,

```
account = BankAccount.new()
puts account.accountNumber
puts account.accountName
```

மேலே உள்ள இரண்டு puts கட்டளைகளும் அணுக்க

செயற்கூறுகள் திருப்பி அனுப்பும் இரண்டு மாறிகளின் மதிப்பை (நமது எடுத்துக்காட்டில் "12345" மற்றும் "John Smith") அச்சிடும்.
இப்போழுது உருபொருள் மாறிகளின் மதிப்பைப்பெற (get) முடியும். அடுத்ததாக உருபொருள் மாறிகளில் மதிப்பைப்பொருத்த (set) setter செயற்கூறுகளை பயன்படுத்தலாம்.

```
class BankAccount
   def account_number
      @account_number
   end

   def account_number=( value )
      @account_number = value
   end

   def account_name
      @account_name
   end

   def account_name=( value )
      @account_name = value
```

```
    end

end
```

இப்போழுது வர்க்கதிற்கு ஒரு உருபொருளை உருவாக்கி, *setter method*-டை பயன்படுத்தி *name* மற்றும் *account_number*-க்கு மதிப்பை பொருத்தலாம். மேலும் *getters*-யை பயன்படுத்தி அதை பெறலாம்,

```
account = BankAccount.new()

account.accountNumber = "54321"
account.accountName = "Fred Flintstone"

puts account.accountNumber
puts account.accountName
```

```
kalarani:intro kalarani$ irb
2.2.0 :001 > class BankAccount
2.2.0 :002?>
2.2.0 :003 >         def account_number
2.2.0 :004?>              @account_number
2.2.0 :005?>         end
2.2.0 :006?>
2.2.0 :007 >         def account_number=( value )
2.2.0 :008?>              @account_number = value
2.2.0 :009?>         end
2.2.0 :010?>
2.2.0 :011 >         def account_name
2.2.0 :012?>              @account_name
2.2.0 :013?>         end
2.2.0 :014?>
2.2.0 :015 >         def account_name=( value )
2.2.0 :016?>              @account_name = value
2.2.0 :017?>         end
2.2.0 :018?>
2.2.0 :019 >      end
 => :account_name=
2.2.0 :020 > account = BankAccount.new
 => #<BankAccount:0x007f9a13253870>
2.2.0 :021 > account.account_number = 54321
 => 54321
2.2.0 :022 > account.account_name = "Fred Flintstone"
 => "Fred Flintstone"
2.2.0 :023 > account.account_number
 => 54321
2.2.0 :024 > account.account_name
 => "Fred Flintstone"
2.2.0 :025 >
```

15.6 ரூபி வர்க்க மாறிகள்:

வர்க்க மாறி (Class variable) என்பது ஒரு *மாறி. அது வர்க்கத்தின் எல்லா உருபொருட்களாலும் பகிர்ந்துக்கொள்ளப்படும். வர்க்க மாறிகளை வர்க்கத்தின் வரையறையில் initialize செய்ய வேண்டும். இவற்றின் பெயருக்கு முன்னர் இரண்டு @ குறியீடுகள்(@@) கொடுக்க வேண்டும்.*

இதை விளக்க, @@interest_rate என்கிற *வர்க்க மாறியை சேர்க்கலாம்.(இதனால்* ஒரே *interest மதிப்பு தான்* எல்லா *bank accounts-ற்கும்)*

```
class BankAccount
      @@interest_rate = 0.2
   def interest_rate
        @@interest_rate
   end

   def account_number
      @account_number
   end

   def account_number=( value )
      @account_number = value
```

```
   end

   def account_name
        @account_name
   end

   def account_name=( value )
        @account_name = value
   end

end
```

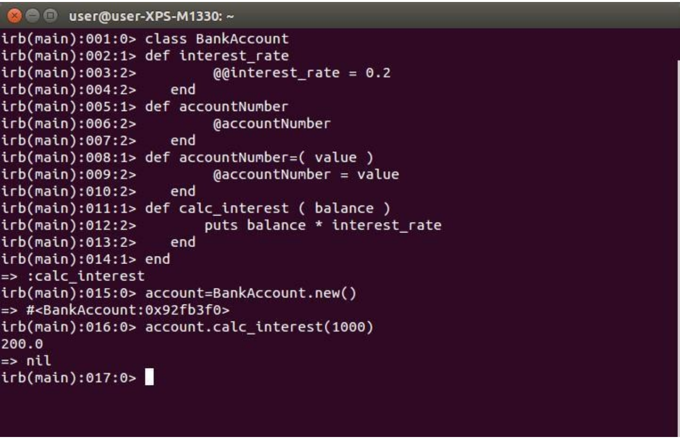
15.7 உருபொருள் செயற்கூறுகள்:

உருபொருள் செயற்கூறுகள் (Instance methods) என்பது *வர்க்கத்தின் உருபொருளைக்கொண்டு அழைக்கப்படும். இவ்வகை செயற்கூறுகள் வர்க்க மாறிகளையும், arguments மதிப்பையும் ஏற்று செயல்பாடிற்கு பயன்படுத்தும். உதாரணத்திற்கு, நமது வர்க்கத்தில் ஒரு புது செயற்கூற்றை உருவாக்கி, அதில் புது account balance-யை argument-ஆக பெற்று, மேலும் @@interest_rate-டை பயன்படுத்தி interest due-வை கணக்கிடலாம்.*

```
def calc_interest ( balance )
       balance * interest_rate
end
```

இப்பொழுது வர்க்கத்திற்கு உருபொருளை உருவாக்கி புதிய செயற்கூற்றை அழைக்கலாம்:

```
account = BankAccount.new()
account.calc_interest 1000
200.0 # 1000 * 0.2
```



```
user@user-XPS-M1330: ~
irb(main):001:0> class BankAccount
irb(main):002:1> def interest_rate
irb(main):003:2>        @@interest_rate = 0.2
irb(main):004:2>    end
irb(main):005:1> def accountNumber
irb(main):006:2>        @accountNumber
irb(main):007:2>    end
irb(main):008:1> def accountNumber=( value )
irb(main):009:2>        @accountNumber = value
irb(main):010:2>    end
irb(main):011:1> def calc_interest ( balance )
irb(main):012:2>       puts balance * interest_rate
irb(main):013:2>    end
irb(main):014:1> end
=> :calc_interest
irb(main):015:0> account=BankAccount.new()
=> #<BankAccount:0x92fb3f0>
irb(main):016:0> account.calc_interest(1000)
200.0
=> nil
irb(main):017:0> 
```

15.8 ரூபி class inheritance:

ரூபி *single inheritance*-யை ஆதரிக்கும். இதில் *subclass* உருவாக்கி மற்றொரு *class*-யிலிருந்து எல்லா உறுப்பினர்களையும் *(variables* மற்றும் *methods) inherit* செய்யலாம். மேலும் *subclass*-ல் *superclass*-யில்லாத புதிய உறுப்பினர்களையும் உருவாக்கலாம்.

ஒரு வர்க்கத்திலிருந்து இன்னொரு வர்க்கத்தை *inherit*

செய்ய < குறியீட்டை பயன்படுத்த வேண்டும். உதாரணத்திற்கு, ஒரு *NewBankAccount* வர்க்கத்தை உருவாக்குவோம். இந்த வர்க்கத்திற்கு அசல் வர்க்கத்திலுள்ள எல்லா உறுப்பினர்களோடு, வாடிக்கையாளரின் தொலைபேசி எண்ணும் தேவை . இதை செய்ய,*BankAccount* வர்க்கத்தை *inherit* செய்து, மேலும் புதிய உருபொருள் மாறியையும் சேர்க்க வேண்டும்.

```
class NewBankAccount < BankAccount

   def customer_phone
        @customer_phone
   end

   def customer_phone=( value )
        @customer_phone = value
   end

end
```

இப்போது *BankAccount* வர்க்கத்திலிருந்து தருவிக்கப்பட்ட *(derived)* ஒரு புதிய வர்க்கம் உள்ளது. இந்த புதிய *subclass*, *superclass*-ல் உள்ள எல்லா உடைமைகளும் உள்ளது மேலும், இந்த வர்க்கத்தில் புதிதாக வாடிக்கையாளரின்

தொலைபேசி எண்ணும் உள்ளது.

```
account.account_number = "54321"
account.customer_phone = "555-123-5433"
54321
555-123-5433
```

```
user@user-XPS-M1330: ~
irb(main):018:0> class BankAccount
irb(main):019:1>     def accountNumber
irb(main):020:2>          @accountNumber
irb(main):021:2>     end
irb(main):022:1>     def accountNumber=( value )
irb(main):023:2>          @accountNumber = value
irb(main):024:2>     end
irb(main):025:1> end
=> :accountNumber=
irb(main):026:0> class NewBankAccount < BankAccount
irb(main):027:1>     def customerPhone
irb(main):028:2>          @customerPhone
irb(main):029:2>     end
irb(main):030:1>     def customerPhone=( value )
irb(main):031:2>          @customerPhone = value
irb(main):032:2>     end
irb(main):033:1> end
=> :customerPhone=
irb(main):034:0> account=NewBankAccount.new()
=> #<NewBankAccount:0x83d0b8c>
irb(main):035:0> account.accountNumber="54321"
=> "54321"
irb(main):036:0> account.customerPhone = "555-123-5433"
=> "555-123-5433"
irb(main):037:0> puts account.accountNumber
54321
=> nil
irb(main):038:0> 
```

மேலே உள்ள எடுத்துக்காட்டில், *BankAccount* வர்க்கத்தின் வரையறையும், *NewBankAccount* வர்க்கத்தின் வரையறையும், ஒரே கோப்பில் உள்ளது. இதே *BankAccount* வர்க்கம் வேறு கோப்பில் இருந்தால் *require* கட்டளையை பயன்படுத்தி *BankAccount* வர்க்கம் உள்ள கோப்பினை உள்ளடக்க *(include)* வேண்டும். *BankAccount* வர்க்கம், "*BankAccount.rb*" கோப்பில் வரையறுக்கப்பட்டுள்ளது என்று வைத்துக்கொண்டால்.

அதை பின்வருமாறு செய்யலாம்.

```
require 'BankAccount'
class NewBankAccount < BankAccount
   def customer_phone
        @customer_phone
   end
   def customer_phone=( value )
        @customer_phone = value
   end
end
```

16 ரூபி Flow Control:

ரூபியின் சக்திவாய்ந்த அம்சங்களில் கட்டுப்படுத்தும் கட்டமைப்புகள் (control structures) ஒன்று. நிரலில் அறிவுதிறத்தையும் (intelligence), தர்க்கத்தையும் (logic) இணைக்க இந்த கட்டமைப்புகள் உதவுகிறது. கட்டுப்படுத்தும் கட்டமைப்புகளை மற்றும் தர்க்க கட்டளைகளைப் பயன்படுத்தி என்ன நிரலை செயல்படுத்தவேண்டும் என்று முடிவு செய்யப்படுகிறது. இவற்றை எப்படி பயன்படுத்த வேண்டும் என்று இந்த அத்தியாயத்தில் பார்ப்போம்.

16.1 ரூபி நிபந்தனை கட்டளை:

ரூபியின் கட்டுப்படுத்தும் கட்டமைப்புகளில், நிபந்தனை கட்டளை (if statement) மிகவும் அடிப்படையானதாகும். நிபந்தனை கட்டளையின் அமைப்பு பின்வருமாறு,

```
if expression then
  ruby code
end
```

மேலேவுள்ள அமைப்பில் expression என்பது தர்க்க கட்டளை ஆகும். அது ஒன்று true-வாகவோ அல்லது false ஆகவோ இருக்கும். Expression true ஆக இருந்தால் ruby

code இயக்கப்படும். இல்லையெனில் அந்த நிரல் இயக்கத்தை தவிர்த்துவிடும். *End* ஆனது நிபந்தனை கட்டளையை முடிவடைய செய்யும்.
ஒரு எடுத்துக்காட்டை பார்ப்போம்:

```
if 10 < 20 then
    print "10 is less than 20"
end
```

இந்த நிரலை செயல்படுத்தினால், "*10 is less than 20*" என்ற சரத்தை *(string)* அச்சிடும். ஏனென்றால் *10<20* என்பது *true* ஆகும்.
மற்ற மொழிகளைப் போல் இல்லாது ரூபியில் பல எளிய முறைகள் உண்டு. அதில் முதலாவதாக, *then* திறவுச்சொல், அதை நீக்கிய பிறகும் ரூபி அதே விடையை தரும்:

```
if 10 < 20
    print "10 is less than 20"
end
```

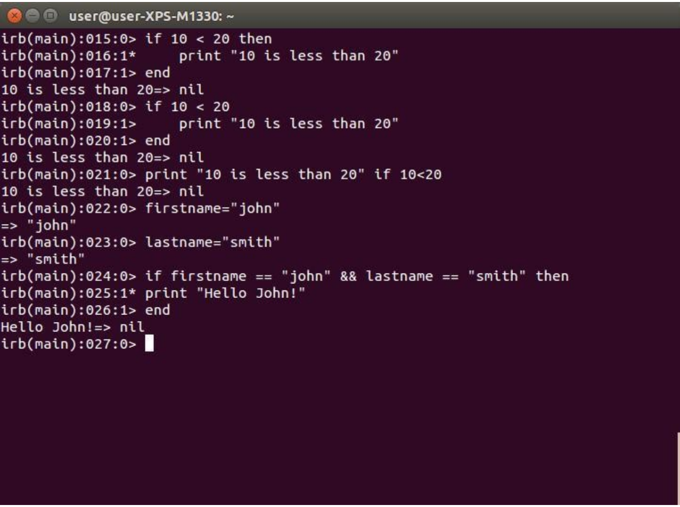
அடுத்ததாக, நிபந்தனை கட்டளையை தொடர்ந்து ஒரே ஒரு வரி நிரல் மட்டும் இருப்பின், *end* திறவுச்சொல்லும் தேவையில்லை . இதனை, பின்வரும் எடுத்துக்காட்டில் காணலாம்.

```
print "10 is less than 20" if 10 < 20
```

ரூபியில் நிரலை சுருக்கமாக எழுதமுடியும் என்பதற்கு இது ஒரு சிறந்த எடுத்துக்காட்டாகும்.

expression-னில் தர்க்க செயற்குறிகளை பயன்படுத்த முடியும். உதாரணத்திற்கு,

```
if firstname == "john" && lastname ==
"smith" then
print "Hello John!"
end
```



16.2else மற்றும் elsif:

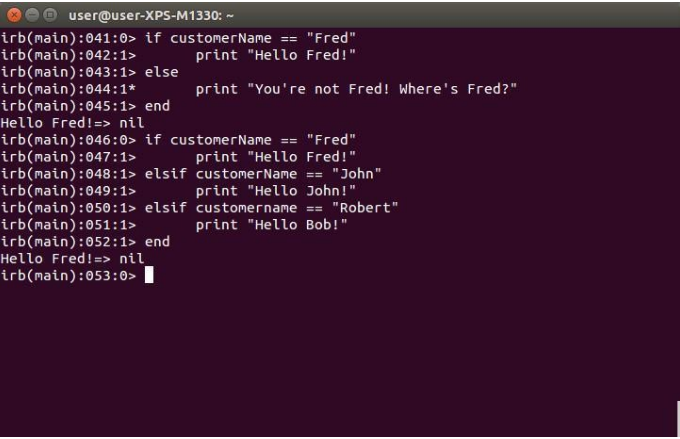
ஒரு குறிப்பிட்ட expression-னை மதீப்பீடு செய்யும்போது true வந்தால் நிபந்தனை கட்டுப்பாட்டு கட்டமைப்பு (if control structure) என்ன செய்யுமென்று பார்த்தோம். மேலும் expression-னை மதீப்பீடு செய்து false வருமாயின் அப்போது if-else கட்டுப்பாட்டு கட்டமைப்பை பயன்படுத்தலாம்.

If-else-ன் அமைப்பு, நிபந்தனை கட்டளையைப் போன்றதுதான் ஆனால் *else* பகுதியும் உண்டு:

```
if customer_name == "Fred"
      print "Hello Fred!"
else
      print "You're not Fred! Where's
Fred?"
end
```

மேலேயுள்ள எடுத்துக்காட்டில் *expression true* ஆகயிருந்தால், நிபந்தனை கட்டளைக்கு அடுத்துள்ள நிரல் பகுதியை செயல்படுத்தும், இல்லையெனில் *else* கட்டளைக்கு அடுத்துள்ள நிரல் பகுதியை செயல்படுத்தும். *else*-ஐ தொடர்ந்து மேலும் ஒரு நிபந்தனை கட்டளை தேவையெனில், *elsif*-ஐ பயன்படுத்தலாம். உதாரணத்திற்கு,

```
if customer_name == "Fred"
      print "Hello Fred!"
elsif customer_name == "John"
      print "Hello John!"
elsif customer_name == "Robert"
      print "Hello Bob!"
end
```



இதே கட்டளையை ";" -னை கொண்டு elsif மற்றும் பதிப்பு கட்டளைகளை (print statement) பிரித்து பயன்படுத்தலாம்.

```
if customer_name == "Fred" ; print "Hello
Fred!"
elsif customer_name == "John" ; print
"Hello John!"
elsif customer_name == "Robert" ; print
"Hello Bob!"
end
```

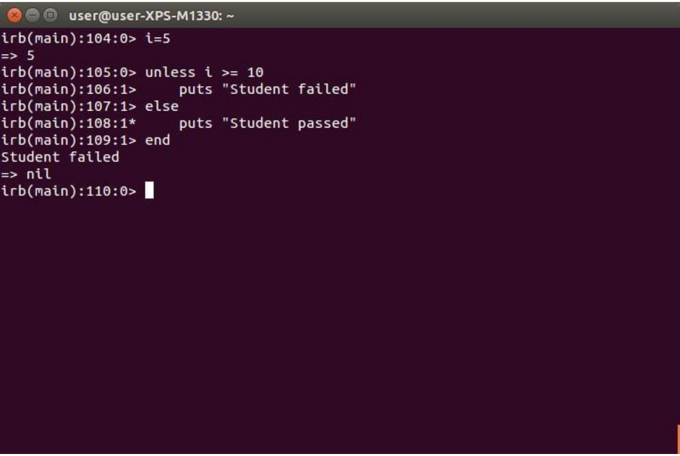
16.3 ரூபி unless statement:

unless கட்டளையானது *if else* பதிலாக பயன்படுத்த கூடிய ஒரு வழியாகும். உதாரணத்திற்கு, *if else* கட்ட்ளையை பயன்படுத்தி பின்வருமாறு எழுதலாம்.

```
if i < 10
   puts "Student failed"
else
   puts "Student passed"
end
```

இந்த நிரலில் *unless* கட்டளையை பயன்படுத்த வேண்டுமெனில் கீழ்கண்டவாறு மாற்றி எழுத வேண்டும்,

```
unless i >= 10
    puts "Student failed"
else
    puts "Student passed"
end
```



16.4 ரூபி ternary செயற்குறி:

ரூபி ternary செயற்குறியைப் பயன்படுத்தி எளிய வழியில் முடிவுகள் எடுக்க முடியும். Ternary செயற்குறியின் அமைப்பு பின்வருமாறு:

[condition] ? [true expression] : [false expression]

மேலே உள்ள அமைப்பில் *[condition]* என்பது ஒரு *expression* ஆகும். அது ஒரு *true*-வாகவோ அல்லது *false*

ஆகவோ இருக்கும். விடை *true*-வாக இருந்தால்*[true expression]* செயல்படுத்தும், விடை *false* ஆக இருந்தால் *[false expression]*-னை செயல்படுத்தும்.உதாரணத்திற்கு,

```
customer_name = "Fred"
=> "Fred"
puts customer_name == "Fred" ? "Hello
Fred" : "Who are you?"
=> "Hello Fred"
```



17 ரூபி case statement:

முந்தைய அத்தியாயத்தில் if...else மற்றும் elsif-யை பயன்படுத்தி சில கட்டுப்பாட்டு கட்டமைப்புகளைப்பற்றி அறிந்துகொண்டோம். இதை கொண்டு ஒரு குறிப்பிட்ட மதீப்பீட்டலே செய்ய முடியும்.(உதாரணத்திற்கு, string மதிப்பை பின்வருமாறு பார்க்கலாம்)

```
if customerName == "Fred"
      print "Hello Fred!"
elsif customerName == "John"
      print "Hello John!"
elsif customername == "Robert"
      print "Hello Bob!"
end
```

நிபந்தனைகளின் எண்ணிக்கை அதிகமாகும்போது if கட்டமைப்பைப் பயன்படுத்துவது கடினமான செயலாகும். இதை எளிதாக கையாள ரூபி case கட்டளையைப் பயன்படுத்தலாம். Case கட்டளையின் அமைப்பு பின்வருமாறு:

```
result = case value

   when match1; result1
```

```
    when match2; result2
    when match3; result3
    when match4; result4
    when match5; result5
    when match6; result6
    else result7
end
```

தேவைக்கேற்ப எத்தனை when கட்டளைகள் வேண்டுமோ இருக்கலாம். Case கட்டளையில் உள்ள options(இதில் match1 முதல் match7 வரை) ஒப்பிட்டு பார்க்கும், அதில் பொருத்தமான விடையை result மாறியில் வைக்கும். எந்த மதிப்பும் பொருந்தவில்லையெனில் else கட்டளையில் உள்ள விடையை result மாறிக்கு அளிக்கும். இதை விளக்க நாம் ஒரு எடுத்துக்காட்டை பார்ப்போம். ரூபியில் case கட்டளையை பயன்படுத்தி ஒரு குறிப்பிட்ட மகிழ்வுந்து மாதிரியின் பெயரை உற்பத்தியாளரின் பெயருடன் ஒப்பிட்டு பார்க்கலாம். பொருந்தும் மாதிரியின் பெயரையும் மற்றும் உற்பத்தியாளரின் பெயரையும் அச்சிடலாம்.

```
car = "Patriot"

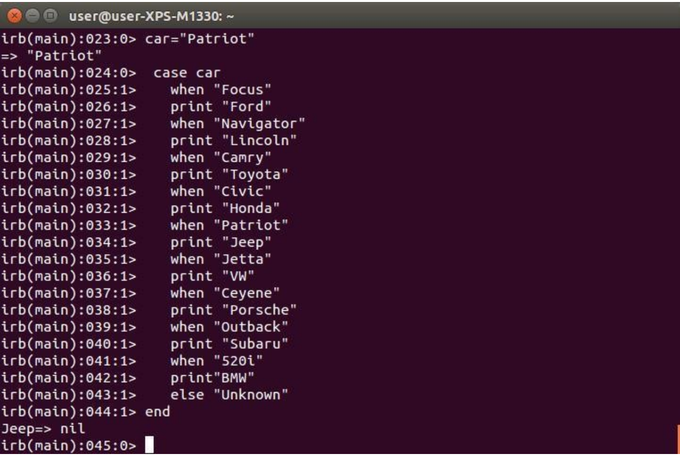
manufacturer = case car
   when "Focus"; "Ford"
   when "Navigator"; "Lincoln"
   when "Camry"; "Toyota"
   when "Civic"; "Honda"
   when "Patriot"; "Jeep"
   when "Jetta"; "VW"
   when "Ceyene"; "Porsche"
   when "Outback"; "Subaru"
   when "520i"; "BMW"
   when "Tundra"; "Nissan"
   else "Unknown"
end

puts "The " + car  + " is made by "  +
manufacturer
```

இதை செயல்படுத்தினால் விடை பின்வருமாறு,

```
The Patriot is made by Jeep
```

மதிப்பு பொருந்தவில்லையெனில் else கட்டளை பின்வரும் விடையை கொடுக்கும்,



```
user@user-XPS-M1330: ~
irb(main):023:0> car="Patriot"
=> "Patriot"
irb(main):024:0>  case car
irb(main):025:1>    when "Focus"
irb(main):026:1>    print "Ford"
irb(main):027:1>    when "Navigator"
irb(main):028:1>    print "Lincoln"
irb(main):029:1>    when "Camry"
irb(main):030:1>    print "Toyota"
irb(main):031:1>    when "Civic"
irb(main):032:1>    print "Honda"
irb(main):033:1>    when "Patriot"
irb(main):034:1>    print "Jeep"
irb(main):035:1>    when "Jetta"
irb(main):036:1>    print "VW"
irb(main):037:1>    when "Ceyene"
irb(main):038:1>    print "Porsche"
irb(main):039:1>    when "Outback"
irb(main):040:1>    print "Subaru"
irb(main):041:1>    when "520i"
irb(main):042:1>    print"BMW"
irb(main):043:1>    else "Unknown"
irb(main):044:1> end
Jeep=> nil
irb(main):045:0>
```

```
The Prius is made by Unknown
```

17.1 எண்களின் Ranges மற்றும் case statement:

Case கட்டளையானது எல்லாவற்றையும் விட எண்களின் *ranges*-ல் இணைந்து பயன்படும்போது மிகவும் பயன் உள்ளதாக உள்ளது.

பின்வரும் எடுத்துக்காட்டில் *case* கட்டளையைப் பயன்படுத்தி பல்வேறு எண்களின் *ranges*-வுடன் காண்போம்.

```
score = 70

result = case score
   when 0..40; "Fail"
   when 41..60; "Pass"
   when 61..70; "Pass with Merit"
   when 71..100; "Pass with Distinction"
   else "Invalid Score"
end

puts result
```

இந்த நிரலை செயல்படுத்தினால், ***"Pass with Merit"*** *என்ற விடையை பெறலாம்.*

user@user-XPS-M1330: ~

```
irb(main):014:0> score = 70
=> 70
irb(main):015:0> result = case score
irb(main):016:1> when 0..40
irb(main):017:1> puts "Fail"
irb(main):018:1> when 41..60
irb(main):019:1> puts "Pass"
irb(main):020:1> when 61..70
irb(main):021:1> puts "Pass with Merit"
irb(main):022:1> when 70..100
irb(main):023:1> puts "Pass with Distinction"
irb(main):024:1> else
irb(main):025:1* puts "Invalid Score"
irb(main):026:1> end
Pass with Merit
=> nil
irb(main):027:0>
```

18 ரூபியில் while மற்றும் until loops:

ஒரு நிரல்பகுதியை மீண்டும் மீண்டும் இயக்கச்செய்ய, மடக்கு கட்டளைகள் (loop statements)பயன்படுகிறது. இந்த அத்தியாயத்தில் while மற்றும் until மடக்கு கட்டளையை பயன்பாடுகளில் எப்படி பயன்படுத்து என்பதை காணலாம்.

18.1 ரூபி while loop:

ரூபி while ஆனது ஒரு குறிப்பிட்ட expression false ஆகும் வரை அந்த loop செயல்படும்.

```
while expression do
... ruby code here ...
end
```

மேலே உள்ளதில், expression என்பது ரூபி expression ஆகும், இது ஒன்று true-வாகவோ அல்லது false ஆகவோ இருக்கும். ruby code here-இதில் செயல்படுத்த வேண்டிய நிரலாகும். முதலில், while திறவுச்சொல்லையடுத்து கொடுக்கப்பட்டுள்ள expression மதிப்பிடப்படும். அதன் மதிப்பு true-ஆக இருந்தால், while-ஐத்தொடர்ந்து கொடுக்கப்பட்ட நிரல்பகுதி செயல்படுத்தப்படும். இந்த நிரல்பகுதி செயல்படுதப்பட்டபின், expression மீண்டும்

மதிப்பிடப்படும். அதன் மதிப்பைப்பொருத்து நிரல்பகுதி மீண்டும் செயல்படுத்தப்படும். மதிப்பு false-ஆக இருந்தால், நிரல்பகுதி செயல்படுத்தப்படாது.

உதாரணத்திற்கு,

```
i = 0
while i < 5 do
   puts i
   i += 1
end
```

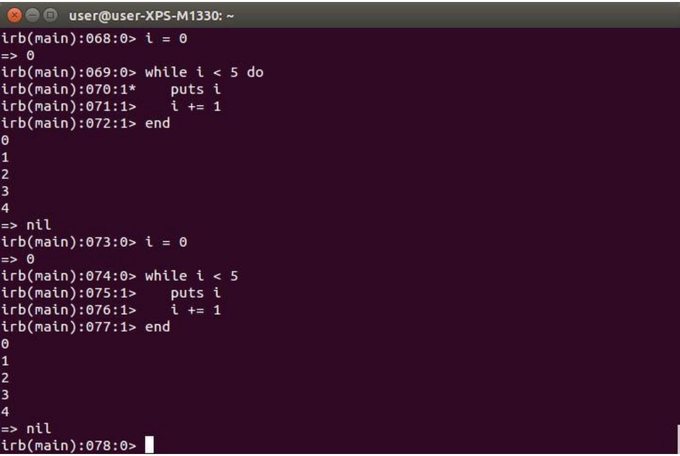
மேலே உள்ள எடுத்துக்காட்டில் i-ன் மதிப்பான 5 விட குறைவாக இருக்கும்வரை,i-ன் மதிப்புகளை அச்சிடும். விடை பின்வருமாறு:

```
0
1
2
3
4
```

இதில் do கொடுப்பது கட்டாயமில்லை,

```
i = 0
while i < 5
   puts i
```

```
    i += 1
end
```



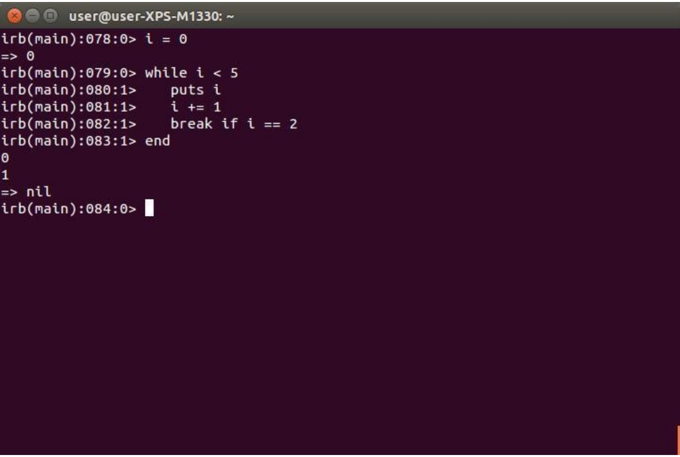
18.2while loops-யை இடைநிறுத்தல்:

சில நேரங்களில் while expression false ஆவதற்கு முன்னதாக while loop-யை இடைநிறுத்தம் செய்ய

நேரிடலாம். இதை *break if statement*-டை கொண்டு செய்யலாம்:

```
i = 0
while i < 5
   puts i
   i += 1
   break if i == 2
end
```

மேலே உள்ள *loop*-யில் *i* ஆனது *5*-க்குப் பதிலாக *2*-ஆக இருக்கும்போதே *loop*-யை விட்டு வெளியேறிவிடும்.



18.3Unless மற்றும் until:

ரூபியின் until கட்டளையை while விட மாறுப்பட்டதாகும். Until expression ஆனது true ஆகும்வரை loop ஆகி கொண்டிருக்கும்.

```
i = 0
until i == 4
   puts i
   i += 1
```

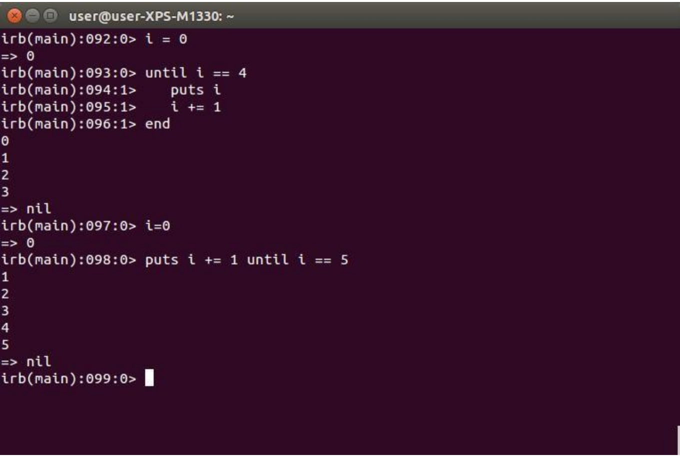
```
end
```

விடை பின்வருமாறு,

```
0
1
2
3
```

Until *கட்டளையை பின்வருமாறும் பயன்படுத்தலாம்,*

```
puts i += 1 until i == 5
```


user@user-XPS-M1330: ~
irb(main):092:0> i = 0
=> 0
irb(main):093:0> until i == 4
irb(main):094:1>   puts i
irb(main):095:1>   i += 1
irb(main):096:1> end
0
1
2
3
=> nil
irb(main):097:0> i=0
=> 0
irb(main):098:0> puts i += 1 until i == 5
1
2
3
4
5
=> nil
irb(main):099:0>

19For loop மற்றும் ரூபியின் looping methods:

முந்தைய அத்தியாயத்தில் ஒரு குறிப்பிட்ட expression true அல்லது false ஆக இருப்பதை கொண்டு ஒரு வேலையை செயல்படுவதை கண்டோம். இந்த அத்தியாயத்தில் for loop, loop,upto,downto மற்றும் times ஆகிய செயற்கூறுகளைக் காணலாம்.

19.1 ரூபியின் for கட்டளை:

For என்ற மடக்கு கட்டளையானது (loop statement) பல நிரலாக்க மொழிகளில் உள்ளது. இது ஒரு குறிப்பிட்ட முறை, ஒரு குறிப்பிட்ட வேலையை தொடர்ந்து செய்யும். உதாரணத்திற்கு,

```
for i in 1..5 do
    puts i
end
```

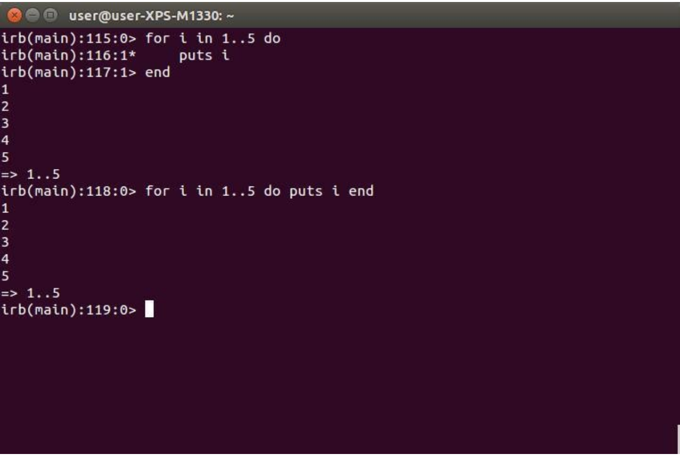
விடை பின்வருமாறு,

```
1
2
3
4
5
```

For கட்டளையில் do என்ற திறவுச்சொல் கட்டாயமானதல்ல. ஆனால் for கட்டளையை ஒரே

வரியில் எழுதினால் do சேர்க்க வேண்டும்:

```
for i in 1..5 do puts i end
```



ரூபியின் ஒரு for கட்டளையை இன்னொரு for கட்டளையின் உள்ளமைப்பாகவும் (nested) தரலாம்,

```
for j in 1..5 do
     for i in 1..5 do
         print i,  " "
     end
puts
```

```
end
```

விடை பின்வருமாறு,

```
1 2 3 4 5
1 2 3 4 5
1 2 3 4 5
1 2 3 4 5
1 2 3 4 5
```

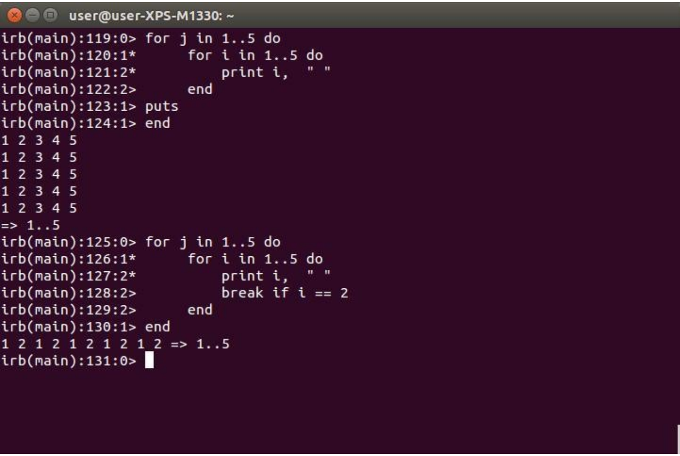
மேலும் for கட்டளையை இடைநிறுத்தம் செய்ய, break if கட்டளையை பயன்படுத்தலாம். இதில் ஒரு விஷயம் கவனிக்க வேண்டும். பல மடக்கு கட்டளைகளை உள்ளடக்கிய ஒரு நிரலில்லிருந்து உள்ளடங்கிய for கட்டளையைவிட்டு வெளியேறினாலும் வெளியிலுள்ள for கட்டளை தொடர்ந்து வேலை செய்யும்:

```
for j in 1..5 do
     for i in 1..5 do
         print i,  " "
         break if i == 2
     end
end
```

i=2 இருக்கும்பொழுதே உள்ளடங்கிய மடக்கு கட்டளை இடையில் நிறுத்தப்பட்டு, நிரலோட்டம் வெளி மடக்கு கட்டளைக்கு சென்று விடும்.

```
1 2
1 2
```

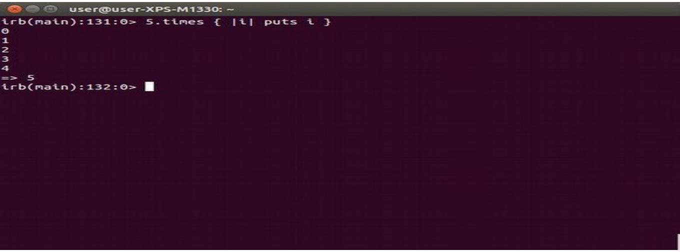
```
1 2
1 2
1 2
```



19.2 ரூபியின் times செயற்கூறு:

Times செயற்கூற்றை for கட்டளைக்கு மாற்றாக பயன்படுத்தலாம். இந்த செயற்கூறு integer class-யில் உள்ளது. இது ஒரு வேலையை குறிப்பிட்ட முறை செயல்படும்.

```
5.times { |i| puts i }
```



மேலே உள்ள உதாரணம் ஆனது பின்வரும் for கட்டளைக்கு இணையானது, மேலும் இதை தட்டச்சிடுவதும் எளிது:

```
for i in 1..5
    puts i
end
```

19.3 ரூபியின் upto செயற்கூறு:

Upto செயற்கூற்றை integer, string மற்றும் date வர்க்கங்களில் பயன்படுத்தலாம். இதை for கட்டளையை போன்று பயன்படுத்த முடியும்.
உதாரணத்திற்கு,

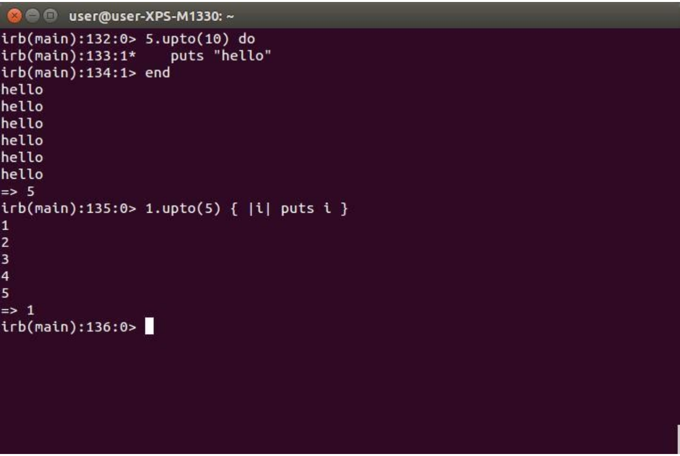
```
for i in 1..5 do
   puts i
end
```

இதற்கு பதிலாக *upto* செயற்கூற்றை பயன்படுத்தலாம். இதில் எத்தனை முறை *loop*-ஆக வேண்டுமோ அதை இந்த செயற்கூற்றின் *argument*-ஆக அனுப்ப வேண்டும்.

```
5.upto(10) do
   puts "hello"
end
```

இதை சுருக்கி ஒரே வரியில் எழுதலாம்,

```
1.upto(5) { |i| puts i }
```



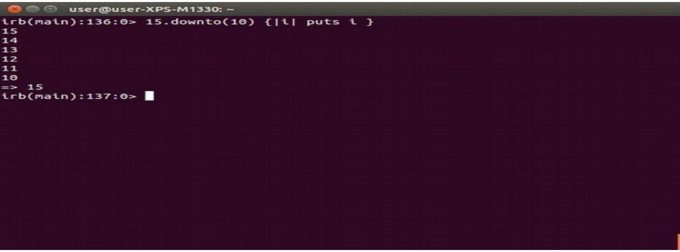
19.4 ரூபியின் downto செயற்கூறு:

Downto செயற்கூறு, *upto* செயற்கூற்றை போன்றதுதான். *upto* செயற்கூறு ஏறுவரிசையில் இயங்கும். *downto* செயற்கூறு இறங்குவரிசையில் இயங்கும்.

உதாரணத்திற்கு:

```
15.downto(10) {|i| puts i }
15
14
13
12
11
```

10


user@user-XPS-M1330: ~
irb(main):136:0> 15.downto(10) {|i| puts i }
15
14
13
12
11
10
=> 15
irb(main):137:0>

20 ரூபி strings:

சரம் (String) என்பது குறியீடுகளின் (characters) குழுவாகும். இது மனிதர்கள் வார்த்தைகள் மற்றும் சொற்றொடர்களை படிக்க உதவுகிறது. சரத்தை கையாளும் பகுதி நிரலாக்கத்தில் ஒரு முக்கிய பகுதியாகும். இந்த அத்தியாயத்தில் சரங்களின் அடிப்படைகளை காண்போம்.

20.1 ரூபியில் சரங்களை உருவாக்குதல்:

ரூபியில் String வர்க்கத்திலிருந்து சரங்களை உருவாக்கலாம். கூடுதலாக, இந்த பொருளில் பல்வேறு செயற்கூறுகள் உள்ளன. இதனை பயன்படுத்தி சரங்களைக் கையாளலாம்.

String வர்க்கத்திலுள்ள new செயற்கூற்றைக் கொண்டு ஒரு புது சரத்தை உருவாக்கலாம்.

```
myString = String.new
=> ""
```

மேலே உள்ள செயற்கூறு ஒரு காலியான சரத்தை உருவாக்கும். மாறாக new செயற்கூற்றில் ஒரு சரத்தை argument-ல் அனுப்பி ஒரு புது சரத்தை உருவாக்கலாம்.

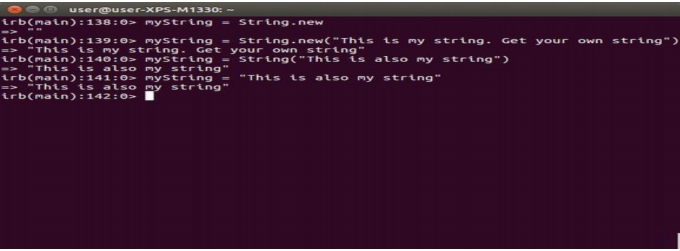
```
myString = String.new("This is my string.
Get your own string")
```

மற்றொரு வழியாக, kernel-லிலுள்ள string செயற்கூற்றை பயன்படுத்தியும் ஒரு சரத்தை உருவாக்கலாம்.

```
myString = String("This is also my
string")
```

ஆனால், எளிய வழியில் சரத்தை உருவாக்க, ஒரு மாறியில் மேற்கோள் குறிகளுக்கிடையே (quotes) கொடுத்தால் போதுமானது. மற்றவற்றை ரூபி பார்த்து கொள்ளும்:

```
myString = "This is also my string"
```



20.2 ரூபி strings-யை quote செய்தல்:

சரங்களை இரட்டை மேற்கோள் குறியிலோ (double quotes(")) அல்லது ஒற்றை மேற்கோள் குறியிலோ (single quotes(')) கொடுக்க முடியும். இரட்டை மேற்கோள் குறியிலுள்ள தப்புவிக்கும் குறியீட்டை (escape character) interpret செய்தால், கொடுக்கப்பட்டிருக்கும் சிறப்பு குறியீட்டிற்கு ஏற்ப, interpreter-ன் வெளியீடு இருக்கும். பின்வரும் உதாரணத்தில் இரட்டை மேற்கோள் குறியின் பயன்பாட்டை பார்ப்போம்:

```
myString = "This is also my string.\nGet
your own string"

puts myString
This is also my string.
Get your own string
```

இதில் \n-என்பது ஒரு புதியவரியை உணர்த்தும் சிறப்பு குறியீடாகும். இதை interpret செய்தால் இரு வரிகளில் சரம் அச்சிடப்படும். இதே மதிப்பு ஒற்றை மேற்கோள் குறியில் மாறுப்பட்ட விடையை தரும்.

```
myString = 'This is also my string.\nGet
your own string'

puts myString
This is also my string.\nGet your own
string
```

```
user@user-XPS-M1330: ~
irb(main):150:0> myString = "This is also my string.\nGet your own string"
=> "This is also my string.\nGet your own string"
irb(main):151:0>
irb(main):152:0* puts myString
This is also my string.
Get your own string
=> nil
irb(main):153:0> myString = 'This is also my string.\nGet your own string'
=> "This is also my string.\\nGet your own string"
irb(main):154:0>
irb(main):155:0* puts myString
This is also my string.\nGet your own string
=> nil
irb(main):156:0>
```

இதில் '\n' அப்படியே அச்சாகும், வேறு எந்த சிறப்பு செயலும் செய்யாது.

20.3 பொதுவான delimited strings:

ரூபி எந்த குறியீட்டை அதன் வரம்பாக (delimiter) பயன்படுத்த வேண்டுமோ அதற்கு முன்னதாக % என்ற குறியீட்டை கொடுத்தால் போதும். உதாரணத்திற்கு, நாம் ampersand(&)-டை வரம்பாக பயன்படுத்தி பார்ப்போம்:

```
myString = %&This is my String&
```

இவ்வாறு வேறொரு குறியீட்டை எல்லையாக பயன்படுத்தும் பொழுது, சரத்தில் உள்ள ஒற்றை மற்றும் இரட்டை மேற்கோள் குறியீடு எல்லைக்குறியீடாக interpreter எடுத்துக்கொள்ளாது.

```
myString = %&This is "my" String&
puts myString
This is "my" String
```

மேலும் parantheses(()), braces({}), square bracket([])-யையும் எல்லைக்குறியீடாக பயன்படுத்தலாம்.

```
myString = %(This is my String)
myString = %[This is my String]
myString = %{This is my String}
```

ரூபி மேலும் சில விசேஷமான எல்லைக்குறியீடுகளைக்கொடுக்கிறது. இரட்டை மேற்கோள் குறிக்கு இணையாக %Q, ஒற்றை மேற்கோள் குறிக்கு %q மற்றும் %x backquote(') —யை எல்லைக்குறியீடுகளாகப்பயன்படுத்தலாம்.
இன்னும் எளிய வழியில், ரூபி சரத்தில் மேற்கோள் குறியை பயன்படுத்த முன்னதாக backslash(\)-யை பயன்படுத்த வேண்டும்.

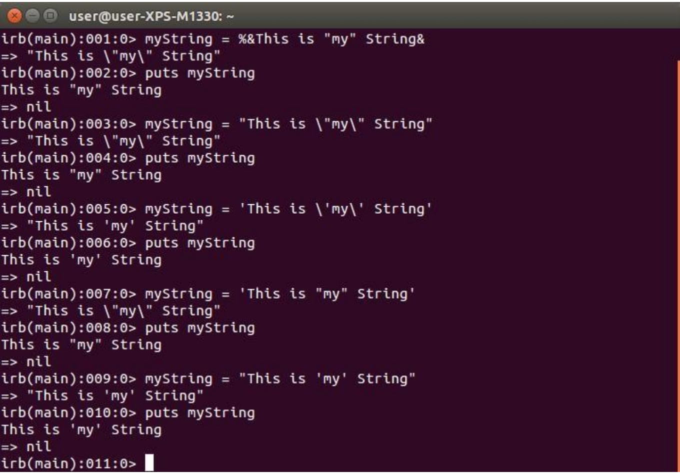
```
myString = "This is \"my\" String"
```

```
myString = 'This is \'my\' String'
```

மற்றொரு வழியாக, தப்புவிக்கும் குறியீட்டைப்பயன்படுத்தவில்லையெனில் இரட்டை மேற்கோள் குறிக்குள், உள்ள் சரத்தில் ஒற்றை மேற்கோள் குறியையும், ஒற்றை மேற்கோள் குறிக்குள் உள்ள சரத்தில் இரட்டை மேற்கோள் குறியையும் பயன்படுத்த வேண்டும்:

```
myString = 'This is "my" String'

myString = "This is 'my' String"
```



```
user@user-XPS-M1330: ~
irb(main):001:0> myString = %&This is "my" String&
=> "This is \"my\" String"
irb(main):002:0> puts myString
This is "my" String
=> nil
irb(main):003:0> myString = "This is \"my\" String"
=> "This is \"my\" String"
irb(main):004:0> puts myString
This is "my" String
=> nil
irb(main):005:0> myString = 'This is \'my\' String'
=> "This is 'my' String"
irb(main):006:0> puts myString
This is 'my' String
=> nil
irb(main):007:0> myString = 'This is "my" String'
=> "This is \"my\" String"
irb(main):008:0> puts myString
This is "my" String
=> nil
irb(main):009:0> myString = "This is 'my' String"
=> "This is 'my' String"
irb(main):010:0> puts myString
This is 'my' String
=> nil
irb(main):011:0>
```

20.4 ரூபி here documents:

Here document (அல்லது heredoc என்றும் அறியலாம்). இது நாம் விரும்பிய வகையில் சரங்களை எழுத உதவுகிறது. இவ்வகை ஆவணத்தை உருவாக்க << தொடர்ந்து ஒரு சொற்றொடரை கொடுக்க வேண்டும். அந்த சொற்றொடரானது ஆவணத்தின் எல்லைக்குறியீடாக இருக்கும். பின்வரும் உதாரணத்தில் நாம் "Doc" என்னும் சரத்தை எல்லைக்குறியீடாகப்பயன்படுத்தியுள்ளோம்:

```
myText = <<DOC
Please Detach and return this coupon with
your payment.
Do not send cash or coins.

Please write your name and account number
on the check and
make checks payable to:

        Acme Corporation

Thank you for your business.
DOC
```

நாம் எவ்வடிவத்தில் சரத்தை எழுதினோமோ, அந்த வடிவமைப்பில் எவ்வித மாற்றமுமின்றி “myText” என்ற சரம் அச்சிடப்படும்.

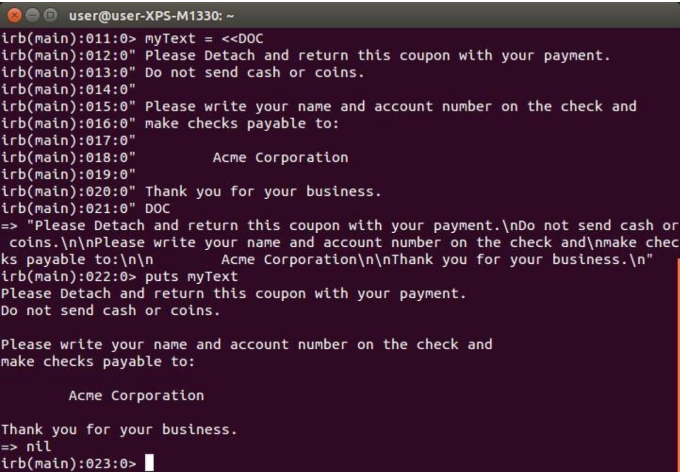
```
puts myText

Please Detach and return this coupon with
your payment.
Do not send cash or coins.

Please write your name and account number
on the check and
make checks payable to:

        Acme Corporation
```

Thank you for your business.



```
user@user-XPS-M1330: ~
irb(main):011:0> myText = <<DOC
irb(main):012:0" Please Detach and return this coupon with your payment.
irb(main):013:0" Do not send cash or coins.
irb(main):014:0"
irb(main):015:0" Please write your name and account number on the check and
irb(main):016:0" make checks payable to:
irb(main):017:0"
irb(main):018:0"         Acme Corporation
irb(main):019:0"
irb(main):020:0" Thank you for your business.
irb(main):021:0" DOC
=> "Please Detach and return this coupon with your payment.\nDo not send cash or
 coins.\n\nPlease write your name and account number on the check and\nmake chec
ks payable to:\n\n        Acme Corporation\n\nThank you for your business.\n"
irb(main):022:0> puts myText
Please Detach and return this coupon with your payment.
Do not send cash or coins.

Please write your name and account number on the check and
make checks payable to:

        Acme Corporation

Thank you for your business.
=> nil
irb(main):023:0>
```

20.5String objects:

ரூபியில் எல்லாமே பொருட்கள் தான் என முந்தைய அத்தியாயத்தில் அறிந்தோம். ஆகவே சரங்களும் பொருட்கள் தான். இந்த சரம் என்னும் பொருளில் பல செயற்கூறுகள் உள்ளன. இவற்றை பயன்படுத்தி சரத்தின் விவரங்களை அறியலாம். உதாரணத்திற்கு சரம் காலியாக

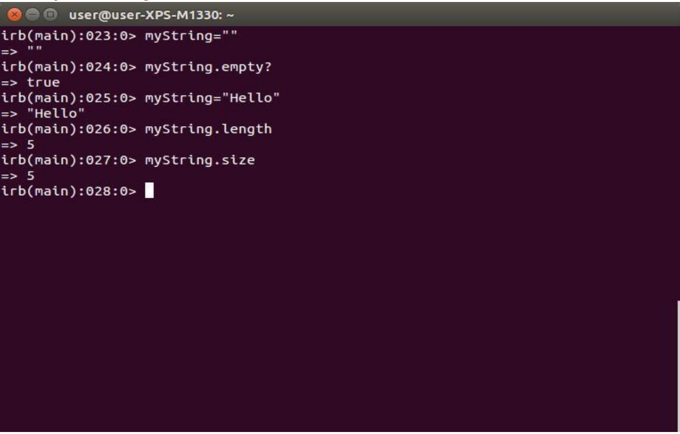
உள்ளதா என்பதை *empty?* என்ற செயற்கூற்றைக் கொண்டு அறியலாம்.

```
myString = ""
=> ""
myString.empty?
=> true
```

மேலும் சரத்தின் நீளத்தையும் அறிய *length* மற்றும் *size* செயற்கூறுகளை பயன்படுத்தி அறியலாம்.

```
myString = "Hello"
myString.length
=> 5
```

```
myString.size
```




```
=> 5
```

21 ரூபியில் சரங்களை இணைத்தல் மற்றும் ஒப்பிடுதல்:

முந்தைய அத்தியாயத்தில் ரூபியில் string வர்க்கத்திற்கு பொருட்களை உருவாக்குவது எப்படி என்று பார்த்தோம். இந்த அத்தியாயத்தில் ரூபியில் சரங்களைப் பெறுதல், ஒப்பிடுதல் மற்றும் இணைத்தலை காண்போம்.

21.1 ரூபியில் சரங்களை இணைத்தல்:

முந்தைய அத்தியாயங்களில் படித்தது போல, ரூபியில் ஒரு வேலையைச் செய்ய பல வழிகள் உள்ளன. அதேப்போல் சரங்களை இணைக்கவும் பல வழிகள் உள்ளன.

'+' செயற்கூற்றை பயன்படுத்தி சரங்களை இணைக்கலாம்:

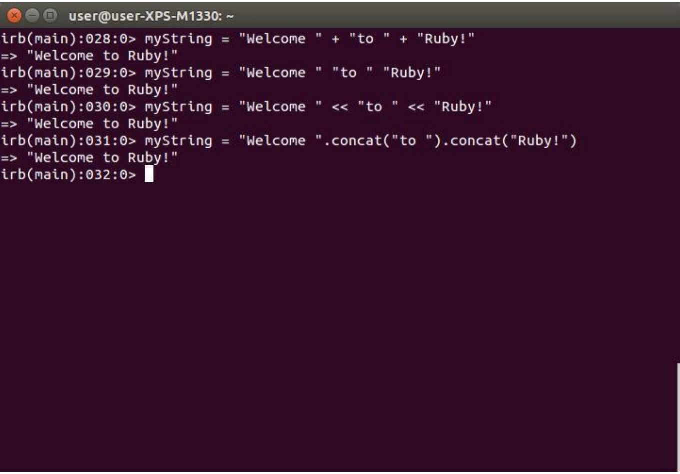
```
myString = "Welcome " + "to " + "Ruby!"
=> "Welcome to Ruby!"
```

மேலும் + குறியீட்டை அகற்றி சரங்களை இணைக்கலாம்:

```
myString = "Welcome " "to " "Ruby!"
=> "Welcome to Ruby!"
```

மேற்குறிப்பிட்ட வழிகள் மட்டுமல்லாது நாம் << method-டை பயன்படுத்தி சரங்களை இணைக்கலாம்:

```
mySțring = "Welcome " << "to " << "Ruby!"
=> "Welcome to Ruby!"
```



மேலும் *concat* செயற்கூற்றை பயன்படுத்தியும் இணைக்கலாம்:

```
myString = "Welcome ".concat("to
").concat("Ruby!")
=> "Welcome to Ruby!"
```

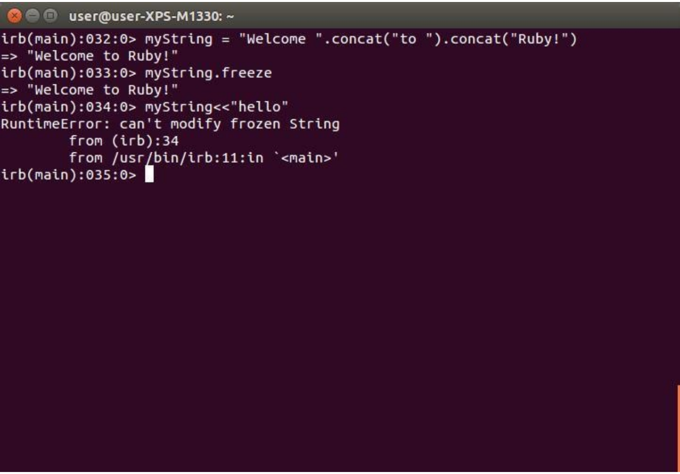
21.2 ரூபியில் சரங்களை உறையவைத்தல்:

ரூபியில் ஒரு சரத்தை உருவாக்கிய பிறகு அதை உறைய வைக்க முடியும். இதனால் நாம் மேலும் அந்த சரத்தை மாற்ற இயலாது. இதை string வர்க்கத்திலுள்ள freeze செயற்கூற்றைக் கொண்டு செய்யலாம்:

```
myString = "Welcome " << "to " << "Ruby!"
=> "Welcome to Ruby!"

myString.freeze

myString << "hello"
TypeError: can't modify frozen string
```



21.3 சரத்தின் கூறுகளை பெறுதல்:

சரத்தின் கூறுகளைப்பெற string வர்க்கத்திலுள்ள [ ] செயற்கூற்றைப் பயன்படுத்தலாம். இந்த செயற்கூற்றைப் பயன்படுத்தி ஒரு குறிப்பிட்ட குறியீடுகளின் தொகுப்பு, சரத்தில் உள்ளதா என்று அறியலாம். குறியீடுகளின் தொகுப்பு சரத்திலிருந்தால், அந்த தொடரை திருப்பியனுப்பும், இல்லையெனில் nil-யை அனுப்பும்.

```
myString = "Welcome to Ruby!"
```

```
myString["Ruby"]
=> "Ruby"

myString["Perl"]
=> nil
```

[ ] செயற்கூற்றில் integer அல்லது ஒரு குறியீட்டின் ASCII code-யை அனுப்பினால் சரத்தில் அந்த இடத்தில் உள்ள குறியீட்டை திருப்பி அனுப்பும். Chr செயற்கூற்றை பயன்படுத்தி குறியீடாக மாற்றலாம்.

```
myString[3].chr
=> "c"
```

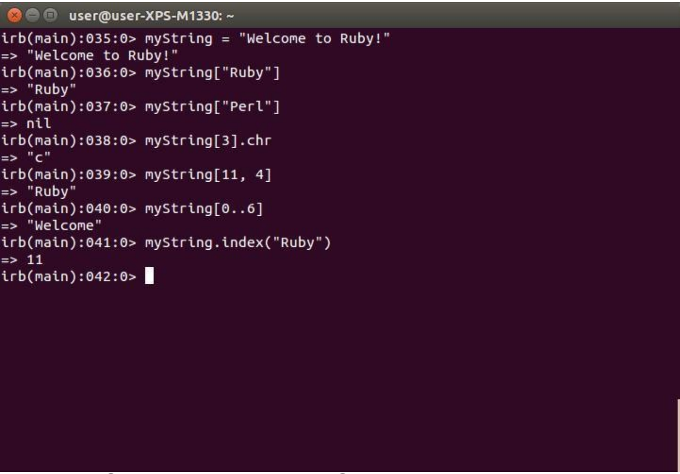
மேலும் ஒரு சரத்திலிருந்து ஒரு குறிப்பிட்ட பகுதியில் உள்ள குறியீடுகளை பெற, ஆரம்ப இடம் மற்றும் பகுதியின் நீளத்தை அனுப்ப வேண்டும்:

```
myString[11, 4]
=> "Ruby"
```

ஒரு range-யை பயன்படுத்தியும் ஒரு குறிப்பிட்ட பகுதியிலுள்ள குறியீடுகளின் குழுவை பெறலாம். ஆரம்பம் மற்றும் முடிவு புள்ளிகளை கொண்டு இந்த பகுதியிலுள்ள குறியீடுகளை அறியலாம்:

```
myString[0..6]
=> "Welcome"
```

பொருந்தும் ஒரு உபசரத்தின் *(Substring)* இடத்தை அறிய



index செயற்கூற்றை பயன்படுத்தி அறியலாம்:

```
myString.index("Ruby")
=> 11
```

21.4 ரூபியில் சரங்களை ஒப்பிடுதல்:

இரண்டு சரங்களை ஒப்பிடுவது ஒரு பொதுவான விசயம். ஒன்று, இரண்டு சரங்கள் சமமாகவோ அல்லது

ஒரு சரம் பெரியதாகவோ அல்லது மற்றதைவிட சிறியதாகவோ இருக்கும்.
சமமானதை அறிய '==' அல்லது eql? என்ற செயற்கூற்றைக் கொண்டு அறியலாம்,

```
"John" == "Fred"
=> false

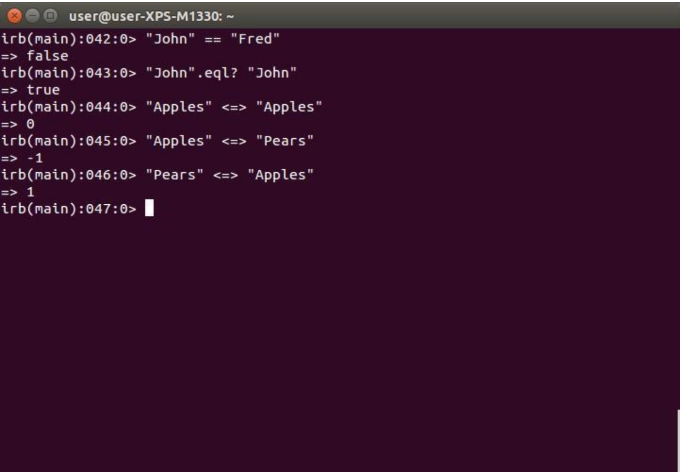
"John".eql? "John"
=> true
```

Spaceship(<=>) method-டை பயன்படுத்தி இரண்டு சரங்களை அகரவரிசையில் ஒப்பிடு செய்யலாம். சரங்கள் சமமாக இருப்பின் ,<=> செயற்கூறு 0-வை திருப்பி அனுப்பும். இடது பக்கம் உள்ள சரம், வலது பக்கம் உள்ள சரத்தை விட சிறியதாக இருந்தால் -1 அனுப்பும். அதுவே பெரியதாக இருந்தால் 1 அனுப்பும்.

```
"Apples" <=> "Apples"
=> 0

"Apples" <=> "Pears"
=> -1

"Pears" <=> "Apples"
=> 1
```



21.5Case insensitive-ஆக string-யை ஒப்பிடுதல்:

சரங்களை பெரிய எழுத்து, சிறிய எழுத்தி வேறுபாடின்றி ஒப்பிட (case insensitive) casecmp என்ற *செயற்கூற்றைப் பயன்படுத்தலாம். இது* <=> *செயற்கூற்றை போலவே 0, -1 அல்லது 1 ஆகிய மதிப்புகளுள் ஒன்றை திருப்பி அனுப்பும்.*

```
“Apples”.casecmp(“APPLES”)
```

```
=> 0

“Apples” <=> “APPLES”
=> 1
```



22 ரூபியில் சரங்களில் மாற்றங்கள் செய்தல்,பொருத்துதல் மற்றும் இடைபுகுத்தல்:

இந்த அத்தியாயத்தில் ரூபியில் சரங்களை மாற்றுதல்,பெருக்குதல் மற்றும் இடைப்புகுத்தலை காணலாம். மேலும், ரூபியின் chomp மற்றும் chop செயற்கூறுகளைப்பற்றியும் காணலாம்.

22.1 சரத்தின் பகுதியை மாற்றுதல்:

ரூபியில் [ ]= செயற்கூற்றை பயன்படுத்தி சரத்தின் பகுதியை மாற்ற இயலும். இந்த செயற்கூற்றைப் பயன்படுத்தி மாற்ற வேண்டிய சரத்தை செயற்கூற்றிற்கு அனுப்பி புதிய சரத்தை அமைக்கலாம். உதாரணம் பின்வருமாறு:

```
myString = "Welcome to JavaScript!"

myString["JavaScript"]= "Ruby"

puts myString
=> "Welcome to Ruby!"
```

இந்த உதாரணத்தில், "JavaScript" என்ற வார்த்தை "Ruby" என்ற வார்த்தை கொண்டு மாற்றி விட்டோம்.

மேற்கண்ட எடுத்துக்காட்டில், ஒரு சொல்லிற்கு பதிலாக, முழுமையாக இன்னொரு சொல்லை சரத்தில் புகுத்தினோம். மாறாக, ஒரு குறியீட்டிற்கு பதிலாக, ஒரு சொல்லை உட்புகுத்தவேண்டுமெனில், *[ ]=* செயற்கூற்றிற்கு, எந்த இடத்தில் மாற்ற வேண்டுமோ அந்த *index*-யை அனுப்பவேண்டும். பின்வரும் உதாரணத்தில், சரத்திலுள்ள ஒரு குறிப்பிட்ட இடத்தில் உள்ள வார்த்தைகளை மாற்றலாம்:

```
myString = "Welcome to JavaScript!"
myString[10]= "Ruby"

puts myString
=> "Welcome toRubyJavaScript!"
```

மேலும் *index range*-யை கொண்டும் மாற்றி அமைக்கலாம். உதாரணத்திற்கு, *index 8* முதல் *20* வரை உள்ள எழுத்துக்களை மாற்றலாம்:

```
myString = "Welcome to JavaScript!"
=> "Welcome to JavaScript!"
```

user@user-XPS-M1330: ~

```
irb(main):052:0> myString = "Welcome to JavaScript!"
=> "Welcome to JavaScript!"
irb(main):053:0> myString["JavaScript"]= "Ruby"
=> "Ruby"
irb(main):054:0> puts myString
Welcome to Ruby!
=> nil
irb(main):055:0> myString = "Welcome to JavaScript!"
=> "Welcome to JavaScript!"
irb(main):056:0> myString[10]= "Ruby"
=> "Ruby"
irb(main):057:0> puts myString
Welcome toRubyJavaScript!
=> nil
irb(main):058:0> myString = "Welcome to JavaScript!"
=> "Welcome to JavaScript!"
irb(main):059:0> myString[8..20]= "Ruby"
=> "Ruby"
irb(main):060:0> puts myString
Welcome Ruby!
=> nil
irb(main):061:0>
```

```
myString[8..20]= "Ruby"
=> "Ruby"

puts myString
=> "Welcome Ruby!"
```

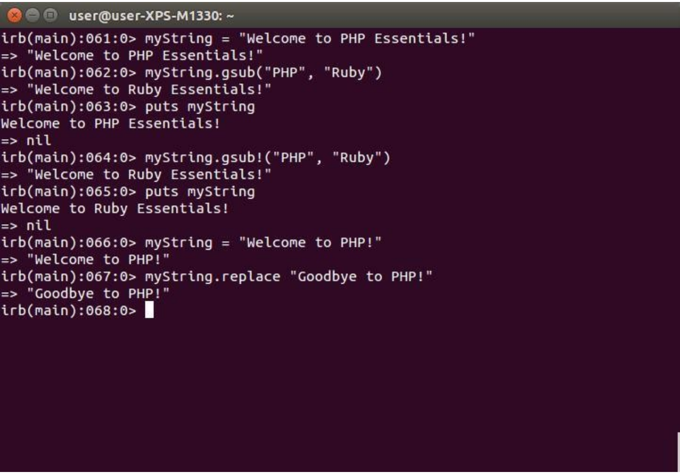
22.2 சரத்தின் ஒரு பகுதியை மாற்றுதல்:

gsub மற்றும் gsub! செயற்கூறுகளைப் பயன்படுத்தி மற்றொரு விரைவான மற்றும் எளிதான வகையில், சரத்தின் ஒரு பகுதியை, மற்றொரு சரத்தைக் கொண்டு மாற்றலாம். இந்த செயற்கூறுகளில் இரண்டு arguments அனுப்ப வேண்டும். அதில் ஒன்று தேடப்படும் சரம் மற்றொன்று புகுத்தப்படவேண்டிய சரம். gsub செயற்கூறு மாற்றப்பட்ட புதிய சரத்தை திருப்பி அனுப்பும். ஆனால் உண்மையான சரத்தில் எந்த மாற்றமும் இருக்காது. மாறாக gsub! செயற்கூறு, நேரடியாக , கொடுக்கப்பட்ட சரத்தையே மாற்றிவிடும். செயற்கூற்றின்ன் இறுதியில் ! இருந்தால், அது அழைக்கப்படும் பொருளில் நேரடி மாற்றங்களைச்செய்யும் என முந்தைய அத்தியாயங்களில் அறிந்த்தை நினைவுகூறலாம்.:

```
myString = "Welcome to PHP Essentials!"
=> "Welcome to PHP Essentials!"

myString.gsub("PHP", "Ruby")
=> "Welcome to Ruby Essentials!"
```

replace *method-டை கொண்டு மொத்த string மாற்றியமைக்க முடியும்:*




```
myString = "Welcome to PHP!"
=> "Welcome to PHP!"

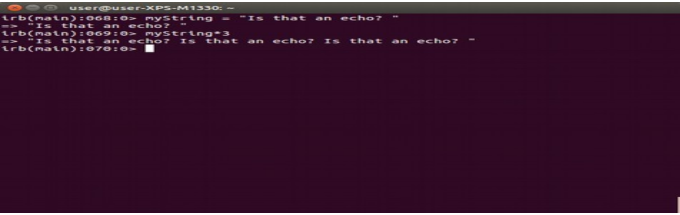
myString.replace "Goodbye to PHP!"
=> "Goodbye to PHP!"
```

22.3 மீண்டும் மீண்டும் ரூபி சரத்தை பதித்தல்:

ஒரு எண்ணைக்கொண்டு, சரத்தை பெருக்க ரூபி அனுமதிக்கிறது. இதற்கு * செயற்கூற்றை பயன்படுத்தலாம். உதாரணமாக, எண் 3-ஐக்கொண்டு, ஒரு சரத்தைப்பெருக்கினால், அந்த சரம் மூன்று முறை அச்சிடப்படுகிறது:

```
myString = "Is that an echo? "
=> "Is that an echo? "

myString * 3
=> "Is that an echo? Is that an echo? Is
that an echo? "
```

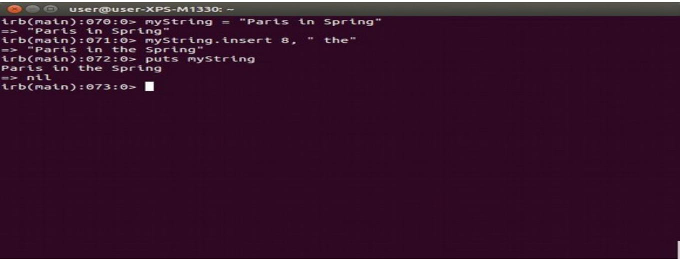


22.4 சரத்தில் சொற்றொடரை இடைப்புகுத்தல்:

இதுவரை இந்த அத்தியாயத்தில் ரூபி சரம் என்ற பொருளிலுள்ள சொற்றொடரை மாற்றியமைப்பதை கண்டோம். மற்றொரு பொதுவான தேவை என்னவென்றால் சரத்தில் ஒரு குறிப்பிட்ட இடத்தில் ஒரு புதிய சொற்றொடரை இடைப்புகுத்தலாம். இதை ரூபியில் insert செயற்கூற்றைக்கொண்டு செய்யலாம். Insert செயற்கூற்றில் arguments-ஆக, எங்கு இடைப்புகுத்த வேண்டுமோ அந்த index-ம் அதை தொடர்ந்து இடைப்புகுத்த வேண்டிய string-யையும் கொடுக்க வேண்டும்:

```
myString = "Paris in Spring"

myString.insert 8, " the"
=> "Paris in the Spring"
```



22.5chomp மற்றும் chop செயற்கூறுகள்:

Chop செயற்கூற்றைக்கொண்டு சரத்திலுள்ள கடைசி எழுத்தை நீக்கலாம்.

```
myString = "Paris in the Spring!"
=> "Paris in the Spring!"

myString.chop
=> "Paris in the Spring"
```

இதில் *chop* செயற்கூறு மாற்றப்பட்ட சரத்தை திருப்பி அனுப்பும். மேலும் இந்த செயற்கூறு கொடுக்கப்பட்ட சரத்தை மாற்றாது. அதற்கு *Chop!* செயற்கூற்றைப் பயன்படுத்த வேண்டும்.

Chomp செயற்கூறு சரத்திலுள்ள *record separators*

நீக்கும். *Record separator $/ variable* கொண்டு வரையறுக்கப்படுகிறது. இயல்பாக *(default)* அது புது



வரிக்குறியீடு *(/n)* ஆகும்.

```
myString = "Please keep\n off the\n grass"
=> "Please keep\n off the\n grass\n"

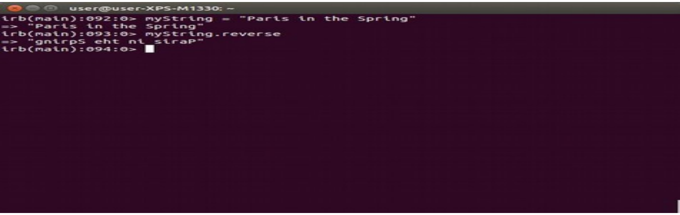
myString.chomp!
=> "Please keep\n off the\n grass"
```

சரத்திலுள்ள எழுத்துக்களை திருப்பி அமைத்தல்:

Reverse செயற்கூற்றைப் பயன்படுத்தி சரத்தின் எழுத்துக்களை திருப்பி அமைக்க முடியும்:

```
myString = "Paris in the Spring"
=> "Paris in the Spring"

myString.reverse
=> "gnirpS eht ni siraP"
```



சரத்தின் case-யை மாற்றுதல்:

சரத்தின் முதல் எழுத்தை ஆங்கிலத்தில் பெரிய எழுத்தாக மாற்ற capitalize மற்றும் capitalize! செயற்கூறுகளைப் பயன்படுத்தலாம்.(முதல் ஒன்று மாற்றப்பட்ட சரத்தை அனுப்பும், இரண்டாவது முதலவதான சரத்தையே மாற்றும்):

```
"paris in the spring".capitalize
```

```
=> "Paris in the spring"
```

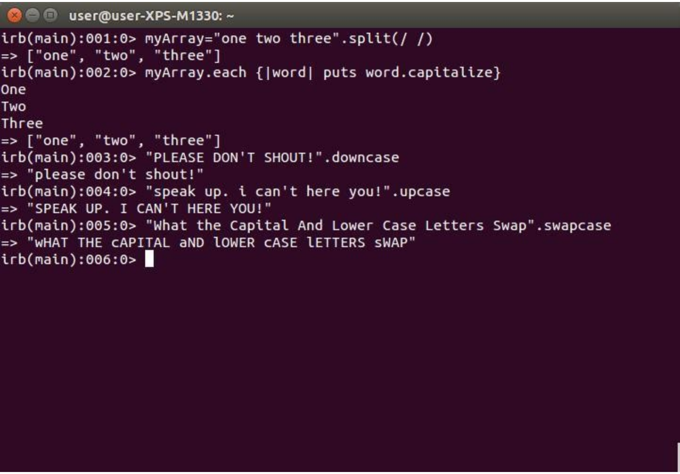
ரூபி சரத்திலுள்ள ஒவ்வொரு வார்த்தையின் முதல் எழுத்தை ஆங்கிலத்தில் பெரிய எழுத்தாக மாற்ற மடக்கு கட்டளைகளைப் பயன்படுத்தலாம்:

```
myArray="one two three".split(/ /)
=> ["one", "two", "three"]
myArray.each {|word| puts word.capitalize}
One
Two
Three
=> ["one", "two", "three"]
```

சரத்திலுள்ள ஒவ்வொரு எழுத்தையும் case மாற்ற upcase,downcase மற்றும் swapcase செயற்கூறுகளைப் பயன்படுத்தலாம். இந்த செயற்கூறுகளை அதன் பெயர் கொண்டே அறியலாம். ஆனால் சந்தேகத்தை நீக்க சில எடுத்துக்காட்டுகளை காணலாம்:

```
"PLEASE DON'T SHOUT!".downcase
=> "please don't shout!"
```

```
"speak up. i can't here you!".upcase
=> "SPEAK UP. I CAN'T HERE YOU!"
```




```
"What the Capital And Lower Case Letters
Swap".swapcase
=> "wHAT THE cAPITAL aND lOWER cASE
lETTERS sWAP"
```

23 சரத்திலுருந்து பிற பொருட்களை உருவாக்குதல்:

இதுவரை சரம் உருவாக்கம்,ஒப்பீடல் மற்றும் கையாளுதல் பார்த்தோம். இந்த அத்தியாயத்தில் சரத்திலிருந்து வேறு வர்க்கத்தை சார்ந்த பொருட்களை எவ்வாறு உருவாக்குவது என காண்போம்.

23.1 சரத்திலிருந்து array-ஐ உருவாக்குதல்:

ஒரு சரத்திலிருந்து array-வைப்பெற split செயற்கூற்றையும் மற்றும் சில செங்கோவைகளையும் (regular expressions) பயன்படுத்த வேண்டும்.
Split செயற்கூறானது சரத்தை பகுதிகளாகப் பிரித்து array கூறுகளாக வைக்கிறது. இந்த மாற்றத்தின்போது split செயற்கூறு எந்த குறியீட்டைப் பயன்படுத்தி பிரிக்க வேண்டும் என்பதை செங்கோவைகள் சொல்கின்றன.
நாம் ஒரு முழுமையான சரத்தை array கூறுகளாக மாற்றுவதை பின்வரும் எடுத்துக்காட்டில் காணலாம்:

```
myArray = "ABCDEFGHIJKLMNOP".split
=> ["ABCDEFGHIJKLMNOP"]
```

இது MyArray என்கிற ஒரு array-க்கான பொருளை உருவாக்கியுள்ளது. எதிர்பாராத விதமாக, இது நமக்கு

பயன்படாது. ஏனென்றால் சரத்திலுள்ள ஒவ்வொரு எழுத்தையும் தனித்தனியான *array* கூறாக வைக்க வேண்டும். இதை செய்ய நாம் செங்கோவைகளைப் பயன்படுத்தவேண்டும். இதில் இரண்டு எழுத்துக்களின் இடையே இருக்கும் புள்ளியாக *(//)* ஒரு செங்கோவையினைக் கொடுக்க வேண்டும். மற்றும் இதை *split* செயற்கூர்றிற்கு *argument* ஆக அனுப்ப வேண்டும்:

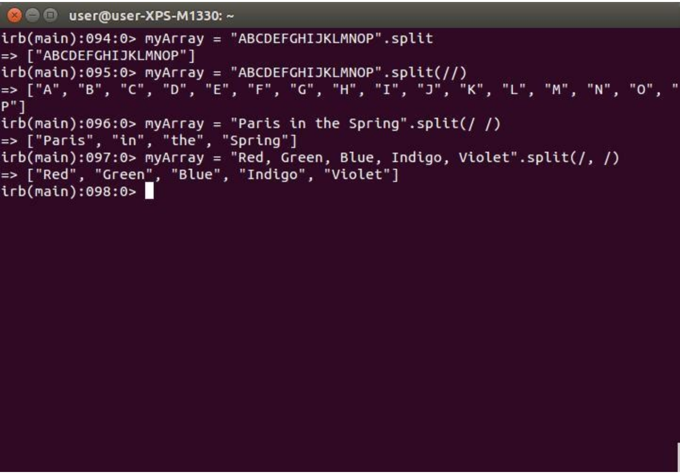
```
myArray = "ABCDEFGHIJKLMNOP".split(//)
=> ["A", "B", "C", "D", "E", "F", "G",
"H", "I", "J", "K", "L", "M", "N", "O",
"P"]
```

மேலும் வார்த்தைகளை அடிப்படையாக கொண்டும் *array*-யை உருவாக்கலாம். இயல்பாகவே *split* செயற்கூறு இரு வார்தைகளுக்கு இடையேயுள்ள இடைவெளியை வைத்து *array* கூறுகளை உருவாக்குகிறது.

```
myArray = "Paris in the Spring".split
=> ["Paris", "in", "the", "Spring"]
```

அல்லது காற்புள்ளியால்(“,”) பிரிக்கப்பட்ட சரத்திலிருந்தும் *array*-வைப்பெறலாம்.

```
myArray = "Red, Green, Blue, Indigo,
Violet".split(/, /)
```




```
=> ["Red", "Green", "Blue", "Indigo",
"Violet"]
```

23.2 சரத்திலிருந்து பிற பொருட்களைப்பெறுதல்:

சரத்திலிருந்து ரூபியிலுள்ள மற்ற வகை பொருட்களையும் (fixnums, floats மற்றும் symbols) பெறலாம்.

சரத்திலிருந்து integer-ஐப்பெற to_i செயற்கூற்றை பயன்படுத்தலாம்:

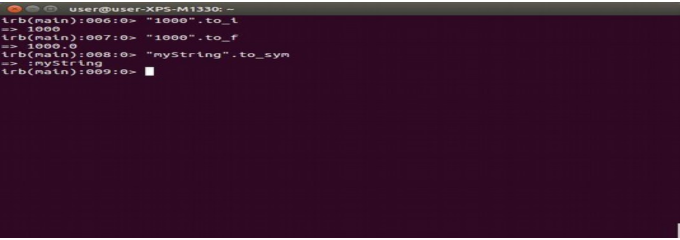
```
"1000".to_i
=> 1000
```

சரத்திலிருந்து floating point-ஐப்பெற to_f செயற்கூற்றை பயன்படுத்தலாம்:

```
"1000".to_f
=> 1000.0
```

சரத்திலிருந்து symbol-ஐப்பெற to_sym செயற்கூற்றை பயன்படுத்தலாம்:

```
"myString".to_sym
=> :myString
```


user@user-XPS-M1330: ~
irb(main):006:0> "1000".to_i
=> 1000
irb(main):007:0> "1000".to_f
=> 1000.0
irb(main):008:0> "myString".to_sym
=> :myString
irb(main):009:0>

24 கோப்பகங்களைக் கையாளுதல்:

இதுவரை ரூபியின் அடிப்படைகளை பார்த்தோம். இந்த அத்தியாயத்தில் ரூபியில் கோப்பு (File) மற்றும் கோப்பகங்கள் (Directory) கையாளுவதை காணலாம்.

24.1 வேறொரு கோப்பகத்திற்கு செல்லுதல்:

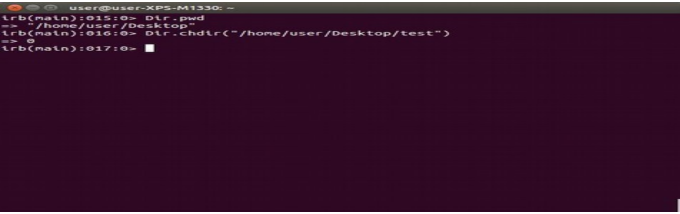
ஒரு குறிப்பிட்ட கோப்பகத்திலிருந்து ரூபி செயலிகளை செயல்படுத்த ஆரம்பிக்கலாம். பெரும்பாலான நேரங்களில், நிரல் மூலமாக, நாம் ஒரு கோப்பகத்திலிருந்து, கோப்பு அமைப்பிலுள்ள (file system) மற்றொரு கோப்பகத்திற்கு போக வேண்டியிருக்கும். ரூபியில் Dir வர்க்கத்தில் பல்வேறு செயற்கூறுகள் உள்ளன. அதை கொண்டு நாம் மற்றொரு கோப்பகத்திற்கு செல்லலாம்.

முதலவதாக நாம் எந்த கோப்பகத்தில் உள்ளோம் என்பதை அறிந்து கொள்வது அவசியமாகும். இதை ரூபியில் Dir வர்க்கத்திலுள்ள pwd செயற்கூற்றைக்கொண்டு அறியலாம்:

```
Dir.pwd
```

ரூபியில் தற்போதைய பயன்பாட்டிலுள்ள கோப்பகத்தை மாற்ற chdir செயற்கூற்றைப் பயன்படுத்தலாம். இந்த செயற்கூற்றில் எந்த கோப்பகத்திற்கு செல்ல வேண்டுமோ அதை argument ஆக கொடுக்க வேண்டும்:

```
Dir.chdir("/home/user/Desktop/test")
```



24.2 புதிய கோப்பகங்களை உருவாக்குதல்:

ஒரு கோப்பகத்தை உருவாக்க ரூபியில் Dir வர்க்கத்திலிருக்கும் mkdir செயற்கூற்றைப் பயன்படுத்தலாம். இந்த செயற்கூற்றில் புதிய கோப்பகத்தின் பாதையை (Path) argument ஆக கொடுக்க வேண்டும்.

```
Dir.mkdir("/home/user/Desktop/temp")
=> 0
```

24.3 கோப்பகத்திலுள்ள உருப்படிகளை பட்டியலிடுதல்:

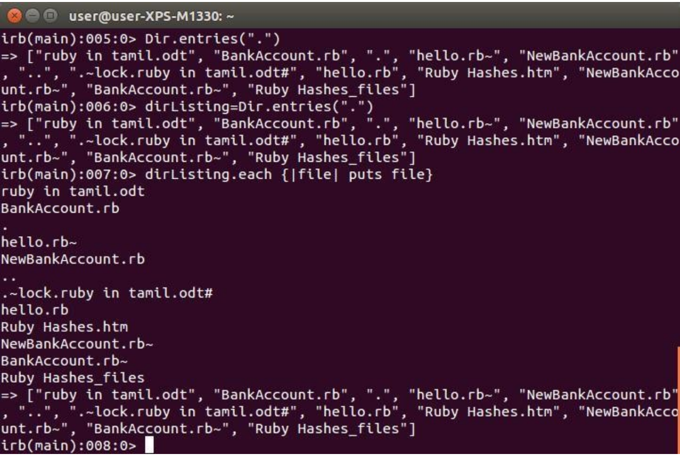
நாம் விரும்பிய கோப்பகத்திற்கு சென்றவுடன், பொதுவான ஒரு தேவை, அதிலுள்ள கோப்புகளைப்பட்டியலிடுதல் ஆகும். இதற்கு *entries method*-ஐப்பயன்படுத்தலாம். *Entries* செயற்கூற்றிற்கு பட்டியலிட வேண்டிய கோப்பகத்தின் பாதையை *argument* ஆக கொடுக்க வேண்டும். அது அந்த கோப்பகத்திலுள்ள கோப்புகளின் பெயரை *array*-யில் திருப்பி அனுப்பும்:

பின்வரும் எடுத்துக்காட்டில், தற்பொழுது உள்ள கோப்பகத்திலுள்ள கோப்புகளின் பட்டியலைக்காணலாம்,

```
Dir.entries(".")
=> ["ruby in tamil.odt", "BankAccount.rb",
".", "hello.rb~", "NewBankAccount.rb",
"..", "~lock.ruby in tamil.odt#",
"hello.rb", "Ruby Hashes.htm",
"NewBankAccount.rb~", "BankAccount.rb~",
"Ruby Hashes_files"]
```

விடையாகப் பெற்ற *Array*-யிலிருந்து அதன் கூறுகளைப்பெற,

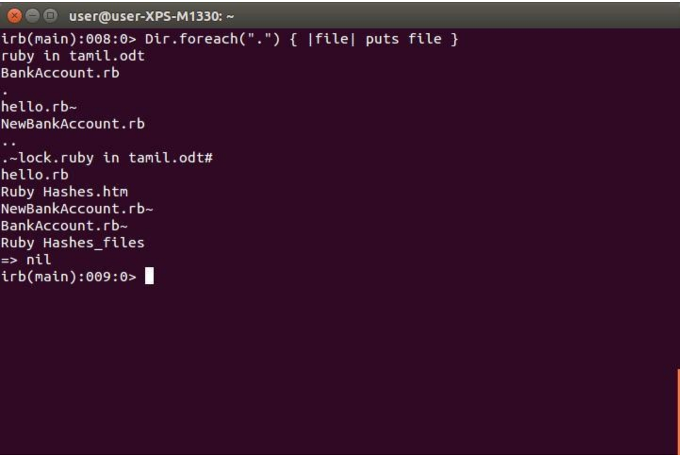
```
dirListing.each { |file| puts file }
```



```
user@user-XPS-M1330: ~
irb(main):005:0> Dir.entries(".")
=> ["ruby in tamil.odt", "BankAccount.rb", ".", "hello.rb~", "NewBankAccount.rb"
, "..", ".~lock.ruby in tamil.odt#", "hello.rb", "Ruby Hashes.htm", "NewBankAcco
unt.rb~", "BankAccount.rb~", "Ruby Hashes_files"]
irb(main):006:0> dirListing=Dir.entries(".")
=> ["ruby in tamil.odt", "BankAccount.rb", ".", "hello.rb~", "NewBankAccount.rb"
, "..", ".~lock.ruby in tamil.odt#", "hello.rb", "Ruby Hashes.htm", "NewBankAcco
unt.rb~", "BankAccount.rb~", "Ruby Hashes_files"]
irb(main):007:0> dirListing.each {|file| puts file}
ruby in tamil.odt
BankAccount.rb
.
hello.rb~
NewBankAccount.rb
..
.~lock.ruby in tamil.odt#
hello.rb
Ruby Hashes.htm
NewBankAccount.rb~
BankAccount.rb~
Ruby Hashes_files
=> ["ruby in tamil.odt", "BankAccount.rb", ".", "hello.rb~", "NewBankAccount.rb"
, "..", ".~lock.ruby in tamil.odt#", "hello.rb", "Ruby Hashes.htm", "NewBankAcco
unt.rb~", "BankAccount.rb~", "Ruby Hashes_files"]
irb(main):008:0>
```

மாற்றுவழியாக, *dir* வர்க்கத்திலுள்ள *foreach* செயற்கூற்றைக் கொண்டு அதே விடையை பெறலாம்:

```
Dir.foreach(".") { |file| puts file }
```


user@user-XPS-M1330: ~
irb(main):008:0> Dir.foreach(".") { |file| puts file }
ruby in tamil.odt
BankAccount.rb
.
hello.rb~
NewBankAccount.rb
..
.~lock.ruby in tamil.odt#
hello.rb
Ruby Hashes.htm
NewBankAccount.rb~
BankAccount.rb~
Ruby Hashes_files
=> nil
irb(main):009:0>

25 ரூபியில் கோப்புகளைக் கையாளுதல்:

முந்தைய அத்தியாயத்தில் கோப்பகங்களை எப்படி கையாளுவதெனப்பார்த்தோம். இந்த அத்தியாயத்தில் ரூபியில் கோப்புகளை எப்படி உருவாக்குவது, எப்படி திறப்பது, படிப்பது மற்றும் எழுதுவது எப்படியென்று பார்ப்போம். மேலும் கோப்புகளை எவ்வாறு அழிப்பது மற்றும் பெயர் மாற்றுவதென்றும் காண்போம்.

25.1 புதிய கோப்பை உருவாக்குதல்:

ரூபியில் ஒரு புது கோப்பை உருவாக்க file வர்க்கத்திலுள்ள new செயற்கூற்றைப் பயன்படுத்தலாம். New செயற்கூற்றிற்கு இரண்டு arguments கொடுக்க வேண்டும். ஒன்று உருவாக்க வேண்டிய கோப்பின் பெயர், மற்றொன்று எந்த முறையில் கோப்பை திறக்க வேண்டும் என்று கொடுக்க வேண்டும்.
கோப்பைத்திறக்கும் முறைகளை பின்வரும் அட்டவணையில் காணலாம்:

Mode	*விளக்கம்*
r	*படிக்க மட்டும் அனுமதி. குறியீட்டு*

	புள்ளியானது கோப்பின் தொடக்கத்தில் இருக்கும்.
r+	படிக்க மற்றும் எழுத அனுமதி. குறியீட்டு புள்ளியானது கோப்பின் தொடக்கத்தில் இருக்கும்.
w	எழுத மட்டும் அனுமதி. குறியீட்டு புள்ளியானது கோப்பின் தொடக்கத்தில் இருக்கும்.
w+	படிக்க மற்றும் எழுத அனுமதி. குறியீட்டு புள்ளியானது கோப்பின் தொடக்கத்தில் இருக்கும்.
a	எழுத மட்டும் அனுமதி. குறியீட்டு புள்ளியானது கோப்பின் முடிவில் இருக்கும்.

a+	படிக்க மற்றும் எழுத அனுமதி. குறியீட்டு புள்ளியானது கோப்பின் முடிவில் இருக்கும்.
b	*Binary File Mode.*மேலே உள்ள செய்முறைகளோடு இணைந்து பயன்படும். *Windows/DOS* மட்டும்.

பின்வரும் எடுத்துக்காட்டில், ஒரு புது கோப்பை எழுதுவதற்கான முறையில் *(write mode)* உருவாக்குவதை காணலாம்:

```
File.new("temp.txt", "w")
=> #<File:temp.txt>
```

ஏற்கனவே உள்ள கோப்பைத் திறத்தல்:

File வர்க்கத்திலுள்ள *open* செயற்கூற்றைக் கொண்டு ஏற்கனவே உள்ள கோப்பை திறக்கலாம்:

```
file = File.open("temp.txt")
=> #<File:temp.txt>
```

ஏற்கனவே உள்ள கோப்பை திறக்க மேலே உள்ள

அட்டவணைப்படி பல செய்முறைகள் உள்ளன. பின்வரும் எடுத்துக்காட்டில், நாம் கோப்பை படிப்பதற்காக (read mode) மட்டும் திறக்க செய்வதை காணலாம்,

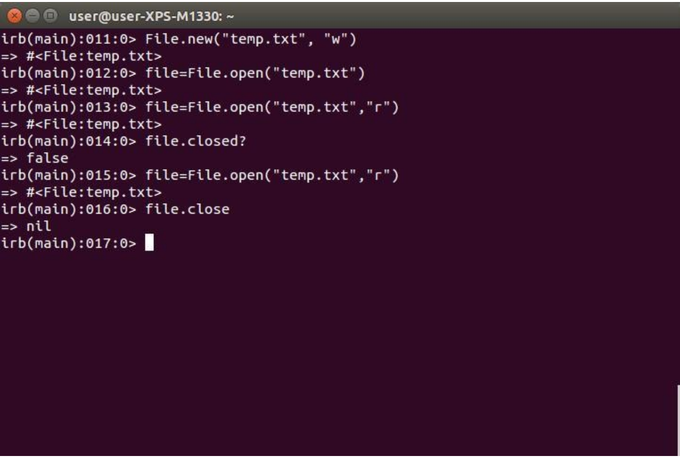
```
file = File.open("temp.txt", "r")
=> #<File:temp.txt>
```

மேலும், கோப்பு ஏற்கனவே திறந்திருக்கிறதா என்பதை closed? என்ற செயற்கூற்றைக் கொண்டு அறியலாம்:

```
file.closed?
=> false
```

முடிவாக, close செயற்கூற்றைக் கொண்டு கோப்பை மூடிவிடமுடியும்:

```
file = File.open("temp.txt", "r")
=> #<File:temp.txt>
file.close
```




```
=> nil
```

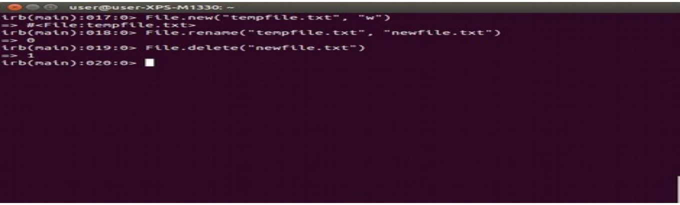
25.2 கோப்பின் பெயரை மாற்றுதல் மற்றும் நீக்குதல்:

ரூபியில் கோப்பை பெயர்மாற்றம் செய்யவும், நீக்கவும் rename மற்றும் delete செயற்கூறுகளை பயன்படுத்தவேண்டும். பின்வரும் உதாரணத்திற்கு ஒரு புதிய கோப்பை உருவாக்கி, பெயர்மாற்றம் செய்து பின்னர், நீக்குவதைக்காணலாம்:

```
File.new("tempfile.txt", "w")
=> #<File:tempfile.txt>
```

```
File.rename("tempfile.txt", "newfile.txt")
=> 0

File.delete("newfile.txt")
=> 1
```



25.3 கோப்புகள் பற்றிய விவரங்களை பெறுதல்:

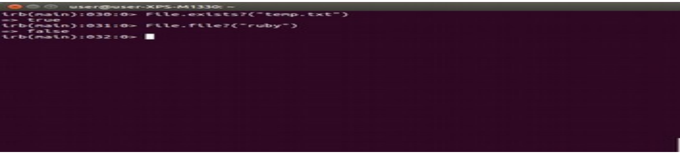
சில நேரங்களில் கோப்பைத்திறப்பதற்கு முன்னதாக அதன் விவரங்களைப்பெறுவது அவசியமாகும். இந்த தேவைக்கேற்ப பல செயற்கூறுகளை File வர்க்கத்தில் காணலாம்:

கொடுக்கப்பட்ட பெயரில் ஒரு கோப்பு இருக்கிறதா என அறிய, exists செயற்கூற்றைப்பயன்படுத்தலாம்.

```
File.exists?("temp.txt")
=> true
```

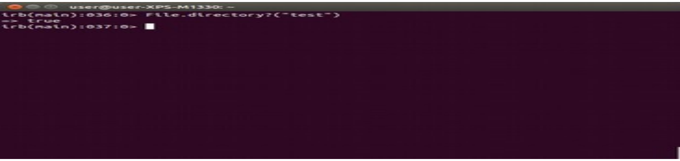
கொடுக்கப்பட்ட பாதையில் உள்ளது ஒரு கோப்புதானா (அல்லது கோப்பகமா) என்பதை, file? செயற்கூறு கொண்டு அறியலாம்.

```
File.file?("ruby")
=> false
```



அதேப்போல், கோப்பகத்தை அறிய directory? செயற்கூற்றைக் கொண்டு கண்டுப்பிடிக்கலாம்:

```
File.directory?("test")
=> true
```

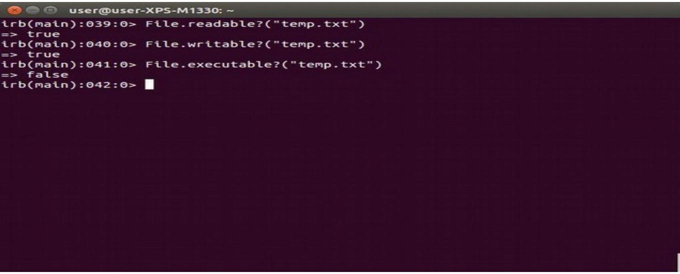


ஒரு கோப்பை படிக்கமுடியுமா, அதில் எழுதமுடியுமா, அதை செயல்படுத்த முடியுமா என்பதை *readable?*, *writable?* மற்றும் *executable?* செயற்கூறுகள் கொண்டு அறியலாம்:

```
File.readable?("temp.txt")
=> true

File.writable?("temp.txt")
=> true

File.executable?("temp.txt")
=> false
```



ஒரு கோப்பின் அளவை அறிய size செயற்கூற்றைப் பயன்படுத்தலாம்:

```
File.size("temp.txt")
=>174
```

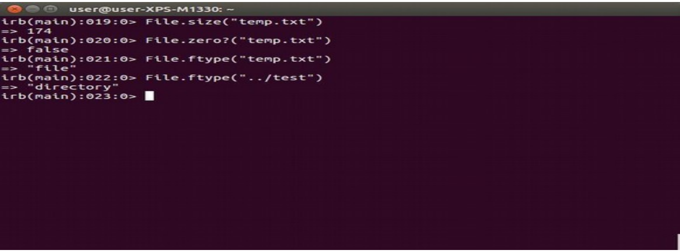
மேலும், கோப்பு காலியாக உள்ளதா என்பதை zero? செயற்கூறு கொண்டு அறியலாம்:

```
File.zero?("temp.txt")
=> false
```

Ftype செயற்கூற்றைக் கொண்டு கோப்பின் வகையை கண்டுப்பிடிக்கலாம்:

```
File.ftype("temp.txt")
=> "file"
```

```
File.ftype("../ruby")
=> "directory"
```

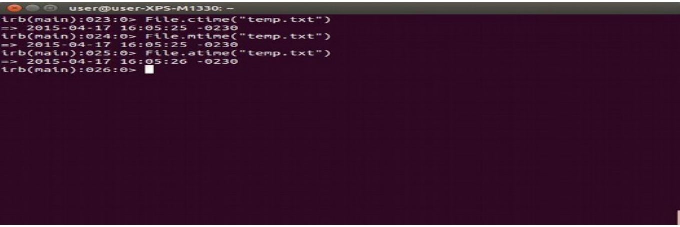


இறுதியாக, கோப்பினை உருவாக்கிய,மாற்றிய மற்றும் பயன்படுத்திய நேரத்தை கண்டுப்பிடிக்க ctime, mtime மற்றும் atime செயற்கூறுகளைக் கொண்டு கண்டுப்பிடிக்கலாம்:

```
File.ctime("temp.txt")

File.mtime("temp.txt")

File.atime("temp.txt")
```



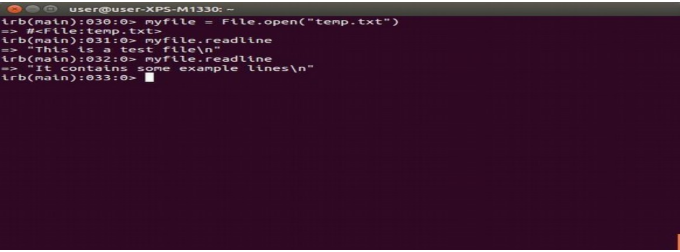
25.4 கோப்பில் எழுத மற்றும் வாசிக்க:

ஒருமுறை, ஏற்கனவே உள்ள கோப்பையோ அல்லது ஒரு புது கோப்பையோ திறந்தால், அதில் நாம் எழுதலாம் மற்றும் படிக்கலாம். நாம் கோப்பைப்படிக்க ஒன்று readline செயற்கூற்றைப் பயன்படுத்தலாம்:

```
myfile = File.open("temp.txt")
=> #<File:temp.txt>

myfile.readline
=> "This is a test file\n"

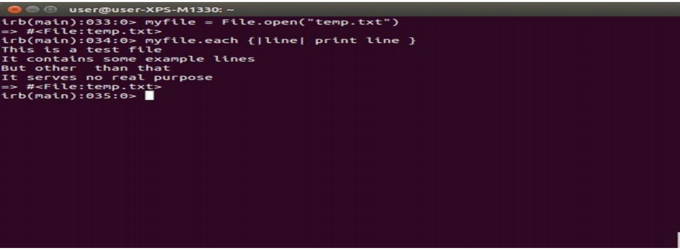
myfile.readline
=> "It contains some example lines\n"
```



மாற்றாக, நாம் each செயற்கூற்றைப் பயன்படுத்தி முழு கோப்பையும் படிக்கலாம்:

```
myfile = File.open("temp.txt")
=> #<File:temp.txt>

myfile.each {|line| print line }
This is a test file
It contains some example lines
But other  than that
It serves no real purpose
```



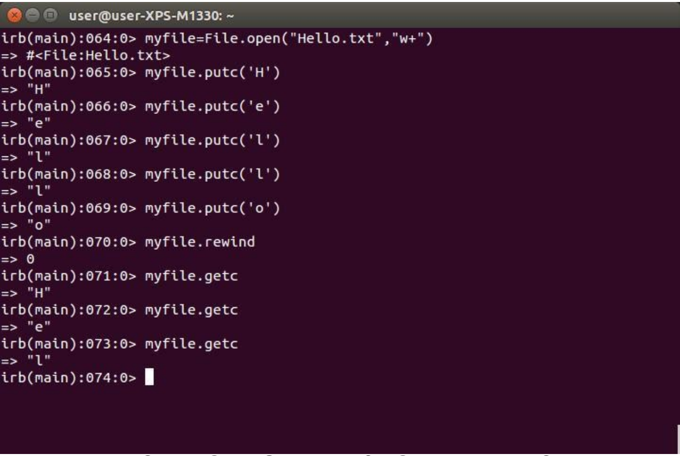
நாம் getc செயற்கூற்றைப் பயன்படுத்தி ஒரு கோப்பில் ஒவ்வொரு எழுத்தாக பெற முடியும்:

```
myfile = File.open("Hello.txt")
=> #<File:temp.txt>

myfile.getc.chr
=> "H"
myfile.getc.chr
=> "e"
myfile.getc.chr
=> "l"
```

Putc செயற்கூற்றைப் பயன்படுத்தி ஒரு நேரத்தில் ஒரு எழுத்தை எழுத முடியும் மற்றும் வார்த்தை, வாக்கியங்களை எழுத puts செயற்கூற்றைப்

பயன்படுத்தலாம். ஆனால் இதில் கவனிக்க வேண்டியது என்னவென்றால் எழுதிய பின் *rewind* செயற்கூற்றை



```
user@user-XPS-M1330: ~
irb(main):064:0> myfile=File.open("Hello.txt","w+")
=> #<File:Hello.txt>
irb(main):065:0> myfile.putc('H')
=> "H"
irb(main):066:0> myfile.putc('e')
=> "e"
irb(main):067:0> myfile.putc('l')
=> "l"
irb(main):068:0> myfile.putc('l')
=> "l"
irb(main):069:0> myfile.putc('o')
=> "o"
irb(main):070:0> myfile.rewind
=> 0
irb(main):071:0> myfile.getc
=> "H"
irb(main):072:0> myfile.getc
=> "e"
irb(main):073:0> myfile.getc
=> "l"
irb(main):074:0>
```

அழைக்க வேண்டும். இது குறியீட்டு புள்ளியை கோப்பின் ஆரம்பத்திற்கு திருப்பி அனுப்பும் அதனால் நாம் எழுதியதை படிக்க இயலும்.

```
myfile = File.new("write.txt", "w+")
=> #<File:write.txt>
```

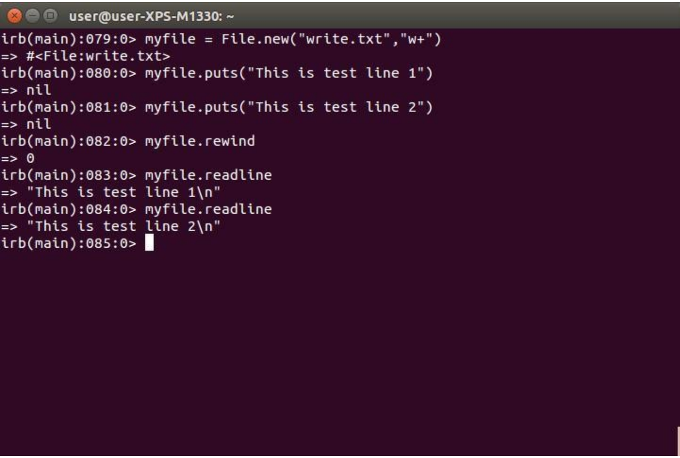
```
myfile.puts("This test line 1")
=> nil

myfile.puts("This test line 2")
=> nil

myfile.rewind
=> 0

myfile.readline
=> "This test line 1\n"

myfile.readline
=> "This test line 2\n"
```



26 முடிவுரை

இந்த நூலில் ரூபி மொழியின் அடிப்படைகளை மட்டுமே பார்த்துள்ளோம்.

இன்னும் இந்த நூலில் எழுதப் படாதவை பல. அவற்றை

வாசகர்கள் இணையத்தில் தேடி, அறிந்து கொள்ள இந்த நூல் ஆர்வத்தைத் தூண்டும் என நம்புகிறேன்.

பின்வரும் இணைப்புகள் மிகவும் பயனுள்ளதாக இருக்கும்.

https://www.ruby-lang.org

http://tryruby.org

https://en.wikibooks.org/wiki/Ruby_Programming

27 கணியம் பற்றி

இலக்குகள்

- *கட்டற்ற கணிநுட்பத்தின் எளிய விஷயங்கள் தொடங்கி அதிநுட்பமான அம்சங்கள் வரை அறிந்திட விழையும் எவருக்கும் தேவையான தகவல்களை தொடர்ச்சியாகத் தரும் தளமாய் உருபெறுவது.*
- *உரை, ஒலி, ஒளி என பல்லூடக வகைகளிலும் விவரங்களை தருவது.*
- *இத்துறையின் நிகழ்வுகளை எடுத்துரைப்பது.*
- *எவரும் பங்களிக்க ஏதுவாய் யாவருக்குமான நெறியில் விவரங்களை வழங்குவது.*
- *அச்சு வடிவிலும், புத்தகங்களாகவும், வட்டுக்களாகவும் விவரங்களை வெளியிடுவது.*

பங்களிக்க

- *விருப்பமுள்ள எவரும் பங்களிக்கலாம்.*
- *கட்டற்ற கணிநுட்பம் சார்ந்த விஷயமாக இருத்தல் வேண்டும்.*
- *பங்களிக்கத் தொடங்கும் முன்னர் கணியத்திற்கு*

உங்களுடைய பதிப்புரிமத்தை அளிக்க எதிர்பார்க்கப்படுகிறீர்கள்.

- *editor@kaniyam.com* முகவரிக்கு கீழ்க்கண்ட விவரங்களடங்கிய மடலொன்றை உறுதிமொழியாய் அளித்துவிட்டு யாரும் பங்களிக்கத் தொடங்கலாம்.
    - **மடலின் பொருள்:** பதிப்புரிமம் அளிப்பு
    - **மடல் உள்ளடக்கம்**
        - என்னால் கணியத்திற்காக அனுப்பப்படும் படைப்புகள் அனைத்தும் கணியத்திற்காக முதன்முதலாய் படைக்கப்பட்டதாக உறுதியளிக்கிறேன்.
        - இதன்பொருட்டு எனக்கிருக்கக்கூடிய பதிப்புரிமத்தினை கணியத்திற்கு வழங்குகிறேன்.
        - உங்களுடயை முழுப்பெயர், தேதி.
- தாங்கள் பங்களிக்க விரும்பும் ஒரு பகுதியில் வேறொருவர் ஏற்கனவே பங்களித்து வருகிறார்

எனின் அவருடன் இணைந்து பணியாற்ற முனையவும்.

- கட்டுரைகள் மொழிபெயர்ப்புகளாகவும், விஷயமறிந்த ஒருவர் சொல்லக் கேட்டு கற்று இயற்றப்பட்டவையாகவும் இருக்கலாம்.
- படைப்புகள் தொடர்களாகவும் இருக்கலாம்.
- தொழில் நுட்பம், கொள்கை விளக்கம், பிரச்சாரம், கதை, கேலிச்சித்திரம், நையாண்டி எனப் பலசுவைகளிலும் இத்துறைக்கு பொருந்தும்படியான ஆக்கங்களாக இருக்கலாம்.
- தங்களுக்கு இயல்பான எந்தவொரு நடையிலும் எழுதலாம்.
- தங்களது படைப்புகளை எளியதொரு உரை ஆவணமாக editor@kaniyam.com முகவரிக்குஅனுப்பிவைக்கவும்.
- தள பராமரிப்பு, ஆதரவளித்தல் உள்ளிட்ட ஏனைய விதங்களிலும் பங்களிக்கலாம்.
- ஐயங்களிருப்பின் editor@kaniyam.com மடலியற்றவும்.

விண்ணப்பங்கள்

- கணித் தொழில்நுட்பத்தை அறிய விழையும் மக்களுக்காக மேற்கொள்ளப்படும் முயற்சியாகும்

இது.

- *இதில் பங்களிக்க தாங்கள் அதிநுட்ப ஆற்றல் வாய்ந்தவராக இருக்க வேண்டும் என்ற கட்டாயமில்லை.*
- *தங்களுக்கு தெரிந்த விஷயத்தை இயன்ற எளிய முறையில் எடுத்துரைக்க ஆர்வம் இருந்தால் போதும்.*
- *இதன் வளர்ச்சி நம் ஒவ்வொருவரின் கையிலுமே உள்ளது.*
- *குறைகளிலிருப்பின் முறையாக தெரியப்படுத்தி முன்னேற்றத்திற்கு வழி வகுக்கவும்.*

வெளியீட்டு விவரம்

அமைத்துக் கொள்ள, தொழில் நோக்கில் பயன்படுத்த அனுமதி வழங்கப்படுகிறது.

ஆசிரியர்: த. சீனிவாசன் — *editor@kaniyam.com +91 98417 95468*

கட்டுரைகளில் வெளிப்படுத்தப்படும் கருத்துக்கள் கட்டுரையாசிரியருக்கே உரியன.